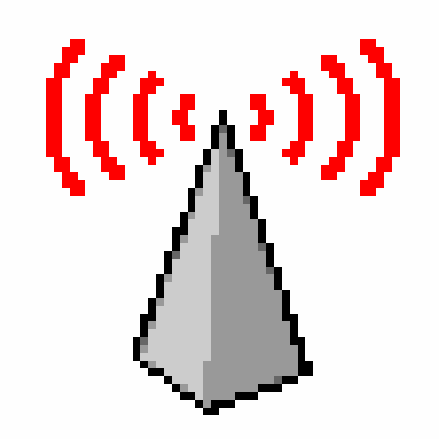
Géolocalisation par WiFi

Damien Larrey, Laurent Rodier

ECOLE DES MINES DE NANCY

Septembre 2005 – Février 2006

Projet scientifique 3A



Géolocalisation par Wi-Fi

Tuteur de projet : Laurent Ciarletta

Elèves : Damien Larrey, Laurent Rodier

Nos remerciements vont exclusivement à M. Ciarletta sans qui nous n'aurions pas atteint l'état actuel du projet. Nous éprouvons beaucoup de respect quant au savoir qu'il nous a transmis et à l'attention qu'il a porté à notre projet. C'est avec un certain regret que nous tournons là, temporairement peut-être, la dernière page d'un travail d'équipe à la fois stimulant et plein d'avenir.

Index

Introduction

Le problème de la géolocalisation a manifestement attiré et mobilisé les esprits de nos ancêtres tout au long de l'histoire. Si autrefois l'on estimait sa position à partir des étoiles et d'instruments comme le polos, le turquet ou le sextant, aujourd'hui les Hommes se sont dotés d'un système de géopositionnement satellitaire bien connu qu'est le GPS. Si les récepteurs les plus simples permettent au plus commun des mortels un positionnement avec une précision inférieure à 100 mètres (moins d'une dizaine de mètres), l'armée américaine ne cache pas d'être capable d'atteindre des précisions inférieures à celle du mètre. Toutefois, le GPS reste un outil spécialisé qui possède un certain nombre de limites et ne peut être vu à ce jour comme un produit de grande consommation. Notre projet scientifique de troisième année a pour ambition d'explorer une récente méthode de localisation basée sur une exploitation passive des réseaux Wi-Fi (Wireless Fidelity), et des caractéristiques physiques de la propagation des ondes radios. Cette méthode est encore en maturation dans les laboratoires de recherche mais ne tardera pas à s'intégrer à l'avenir dans des systèmes embarqués que nous côtoyons tous les jours.

I. Définition du projet

A. *Contexte et motivations*

Avec l'explosion en France du nombre d'usagers connectés à l'ADSL, principalement en zone urbaine, il n'est plus rare maintenant de vouloir bénéficier des avantages du sans-fil Wi-Fi et cela en investissant raisonnablement dans un point d'accès. Cela est d'autant plus vrai que désormais la plupart des assistants personnels (PDAs) et ordinateurs portables sont équipés ne standard de Wi-Fi. Pour se convaincre de l'ampleur du phénomène, une rapide exploration des « airs » à l'aide d'une carte Wi-Fi montre que nous sommes souvent à portée de plusieurs points d'accès. Par ailleurs, on notera qu'il est possible de détecter la présence de points d'accès aussi bien en étant en milieu extérieur qu'intérieur, ce qui n'est pas le cas des signaux GPS. Il n'est donc pas illusoire d'imaginer qu'un dispositif de localisation par Wi-Fi s'appuyant sur un réseau de points d'accès sans-fil puisse fournir une estimation de position précise dans un environnement aussi bien extérieur qu'intérieur. Ceci suppose évidemment que la densité de points d'accès soit suffisamment importante. Dans de telles conditions, ce système viendrait alors remplacer le GPS qui est d'une part incapable de fournir une position

lorsque l'utilisateur est situé dans un bâtiment, et d'autre part peu précis dans des villes où la visibilité sur le ciel est réduite par de hauts immeubles. Enfin, d'un point de vue purement financier, il ne fait aucun doute que l'investissement dans une centaine de point d'accès n'a pas l'ampleur de celui effectué pour le lancement de satellites, surtout si l'on prend en compte les capacités originales des points d'accès (transmission de données bidirectionnelles).

B. *Etat de l'art*

Jusqu'à présent les projets de recherche dans le domaine de la localisation ont exploré une vaste gamme de types de capteurs. Des expériences ont été menées sur des réseaux de capteurs infrarouges, de capteurs vidéo, de capteurs à ultrasons, voire de capteurs magnétiques. Malheureusement ces réseaux sont coûteux à mettre en place et n'offrent pas toujours la précision théorique escomptée. A titre d'exemple, les méthodes optiques souffrent de limitations dues à la portée de vue ou au mauvaises estimations dans des environnements trop exposés aux lumières fluorescentes ou à la lumière solaire. A l'heure actuelle, les chercheurs portent principalement leur attention sur les capteurs radio de type Wi-Fi dont le rapport coût de revient sur portée du signal serait avantageux pour envisager une solution grand public.
Sur la base de nos recherches bibliographiques, nous pouvons décomposer les méthodes de localisation par Wi-Fi en deux grandes familles selon la grandeur physique que l'on mesure : soit une puissance, soit un temps.

1. Localisation par mesure de puissance du signal

Dans la définition du standard 802.11 donnée par l'IEEE intervient la notion de puissance du signal. Intuitivement il nous vient rapidement à l'esprit qu'il s'agit d'une notion importante et nécessaire. En effet, d'un point de vue logiciel, il est indispensable à un moment d'être capable de décider si oui ou non l'environnement est propice à la transmission des données. Etant donné qu'une carte Wi-Fi est capable de fournir une mesure de la puissance du signal à la réception d'un paquet, les chercheurs ont vu là le moyen d'estimer à quelle distance ils étaient du point d'accès émetteur. A travers les exemples de projet décrits ci-dessous, nous essayerons de donner un aperçu des grandes méthodes employées en localisation par mesure de puissance du signal

L'un des premiers travaux sur le sujet est le projet RADAR [1] issu des laboratoires de recherche de Microsoft. Par un système basé sur les ondes radio, les chercheurs démontrent qu'il est possible d'obtenir une estimation de position précise en milieu intérieur sans qu'il soit nécessaire de déployer une infrastructure lourde en capteurs. L'idée du projet consiste à « s'appuyer » sur un réseau local sans fil 802.11 constitué en l'occurrence de plusieurs point d'accès. Durant une première phase appelée phase *offline*, il s'agit de constituer une carte d'empreintes des puissances du signal en enregistrant en divers point de l'espace choisi, un vecteur $(s_1 , \ldots , s_n)$ où s_i est l'intensité du signal en provenance du point d'accès i. Lors d'une seconde phase qualifiée d'*online*, l'estimation de position s'effectue en comparant le vecteur des signaux reçu à un instant t avec ceux enregistrés en phase *offline*. Plus précisément, en supposant une norme euclidienne sur l'espace des signaux, il est possible de déterminer lequel des points de mesures offline est le plus proche du vecteur mesuré. Un algorithme connu sous le nom de « *Multiple nearest neighbors* » permet d'obtenir un gain en précision. Il consiste à déterminer les k point de mesures les plus proches dans l'espace des signaux et de faire ensuite la moyenne des coordonnées spatiales de ces k points. L'inconvénient majeur du système est qu'il n'intègre pas efficacement les variations du signal au cours du temps. Cela peut déboucher sur une fluctuation des estimations parfois incontrôlable et conduisant à des aberrations. Ajoutons aussi que la constitution d'une carte des signaux demande un temps non négligeable lorsqu'on souhaite couvrir correctement une zone. Malgré cela avec une résolution moyenne comprise entre 2 et 3 mètre, le projet RADAR a donné naissance à des techniques d'amélioration (par exemple, ajout d'un algorithme de Viterbi) mais aussi à de toutes nouvelles méthodes.

A l'instar du système GPS, des scientifiques ont en effet cherché à appliquer le principe de la triangulation. Deux différences majeures avec le projet RADAR sont : d'une part la nécessité de disposer d'un modèle de propagation théorique du signal en fonction de la distance au point d'accès, d'autre part de connaître la position des points d'accès avec lesquels on travaille. Le principe est à première vue simple. Etant donné trois mesures de signal en provenance de trois points d'accès différents, on déduit du modèle de propagation une estimation de la distance séparant l'utilisateur des points d'accès en question. L'estimation de la position se fait de manière géométrique en calculant le point d'intersection de trois cercles. Malheureusement les modèles de propagations des ondes en milieu intérieur ne sont pas suffisamment avancés pour tenir compte des perturbations générées par le déplacement des personnes à l'intérieur du bâtiment ou par la présence d'autres obstacles ou par les

phénomènes d'atténuation et de réflexion des ondes. Des équations du signal ont essayé de tenir compte de l'atténuation d'une onde radio lors de la traversée d'un mur, mais de par la trop grande diversité des matériaux composant les bâtiments, il devient insensé de vouloir tout prédire à l'aide d'équations. N'oublions pas non plus qu'un autre phénomène, celui de réflexions multiples d'une onde affecte aussi la précision de mesure du signal. Les méthodes basées sur l'usage de modèles empiriques ou sur l'extrapolation de mesures expérimentales semblent donc temporairement délaissées par les équipes même s'il s'agit d'un objectif qui reste fortement souhaitable .

Le point commun des méthodes que nous venons de voir est l'environnement dans lequel elles sont mises en oeuvre. Nous allons voir à présent un projet qui se veut être une première exploration de la localisation par Wi-Fi en milieu extérieur. Il s'agit du projet collaboratif PlaceLab dirigé par une équipe formée de chercheurs de l'Université de Californie de San Diego et des laboratoires d'Intel Corporation. Le projet reprend le principe de mesurer une empreinte signal par position. L'intérêt de la démarche se situe au niveau des comparaisons de précision selon l'algorithme utilisé (k-nearest neighbors, triangulation, filtre à particules) et de la méthode de capture des données. Pour parvenir à couvrir une zone urbaine, les auteurs ont pratiqué une activité très convoitée aux Etats-Unis et à laquelle on se réfère par l'expression de « War Driving ». Cette activité consiste à relever toutes les puissances des signaux disponibles en un point x donné au moyen du couplage entre GPS, carte Wi-FI et ordinateur portable. Les résultats obtenus révèlent une précision moyenne de l'ordre de 10 mètres même si celle-ci reste sujette à des dégradations lorsque l'utilisateur s'éloigne progressivement du centre ville (diminution de la densité de point d'accès). La précision n'est certes pas encore au rendez-vous même si elle nous parait déjà amplement suffisante pour un bon nombre d'applications urbaines. L'un des principaux atout de PlaceLab est qu'il met à disposition une API libre de droit qui fait des émules et attire des projets comme Active Campus (système de localisation sans-fil à l'Université de Californie de San Diego) ou donne vie à des applications insolites, comme par exemple le couplage de la localisation sans-fil avec des messageries instantanées comme MSN.

Pour terminer, voyons comment les travaux issus du monde de la robotique ont contribué à l'amélioration des systèmes de localisation sans fil. Les meilleurs algorithmes actuels sont pour la plupart fondés sur des modèles probabilistes et plus précisément les filtres de Bayes. Grâce à de tels filtres il est possible de déterminer l'état d'un système dynamique à partir

d'observations bruitées. En théorie cette solution répond exactement aux problèmes sous-jacents à la localisation d'un utilisateur mobile capable de mesurer la force d'un signal Wi-Fi sujet à d'éventuelles variations temporelles. Le filtrage Bayésien est un processus itératif qui repose sur trois étapes majeures : la mise à jour de la distribution de probabilité (en anglais, fonction *Belief State*, symbole : $Bel(x_t)$), la prédiction et la correction. C'est un processus qui permet de prendre en compte le passé. Nous entrerons plus dans les détails lorsqu'on abordera l'algorithme utilisé pour le projet. On divise typiquement la famille des filtres Bayésiens en deux groupes ; d'un côté les filtres continus, de l'autre les filtres discrets.

Dans la première catégorie citons par exemple le filtre de Kalman que l'on utilise souvent avec des réseaux de capteurs vidéo, laser ou pour améliorer les systèmes GPS. La force d'un tel filtre réside surtout dans son efficacité et rapidité de traitement. Appliqué à notre cas, le filtre fournirait une estimation à chaque instant *t* de la position ainsi qu'une estimation de la précision de cette estimation. Les hypothèses associées aux filtres de Kalman font qu'ils ne peuvent générer d'estimations que sur des distributions de probabilité gaussiennes unimodales ; c'est d'ailleurs la grande faiblesse qui leur est reprochée.

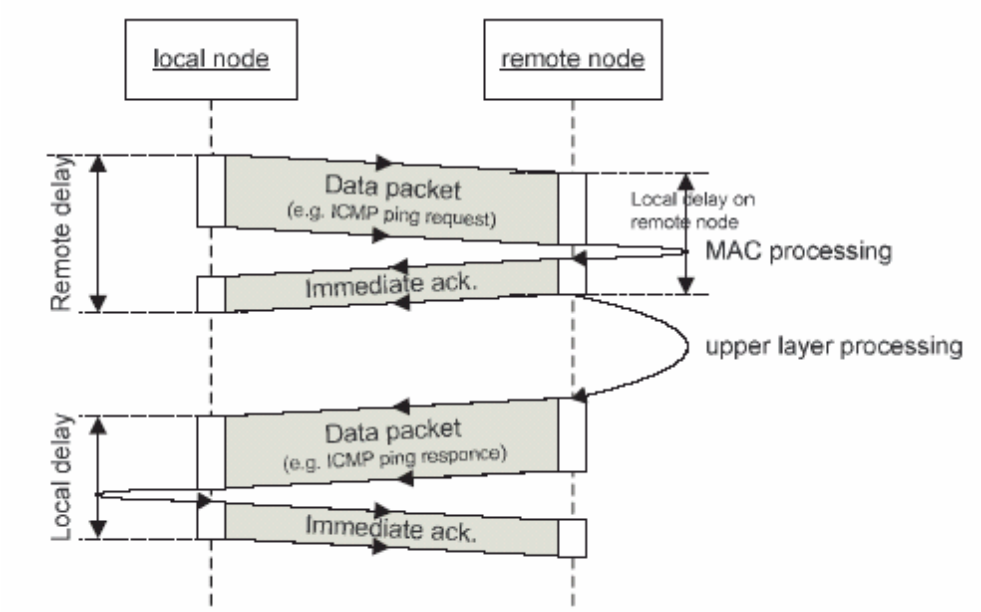
Concernant les filtres discrets, citons tout d'abord les filtres d'approche par grille. L'idée consiste à fractionner tout l'espace sur lequel on travaille en cellules de tailles constantes. Chacune zone contient alors la probabilité de présence de l'utilisateur qui la concerne. L'avantage majeur d'une telle méthode est qu'elle permet la représentation de toutes sortes de distributions de probabilités sur les états de l'espace. En revanche le coût de calcul associé ne lui permet pas de traiter des problèmes sur des espaces de dimension trop élevée. Elle s'avère suffisante pour des problèmes de localisation spatiale à deux voire trois variables. Terminons enfin ce descriptif par la présentation des filtres à particules, eux aussi discrets. Leur principe repose sur une représentation de la distribution de probabilité sous la forme d'un ensemble de particules portant un certain poids. Le poids représente la probabilité de trouver la particule à la position qu'elle occupe. Ce poids intègre « l'historique » de la particule mais il peut aussi tenir compte de la structure du bâtiment (coefficient venant diminuer la probabilité lorsque la particule traverse un mur par exemple). La phase de mise à jour du filtre inclut un procédé de ré-échantillonnage. Comparé respectivement au filtre de Kalman et aux filtres par grille, le filtre à particules permet de travailler avec tout genre de distribution et de se concentrer rapidement sur les zones à fortes probabilités. Pour obtenir des résultas convenables il faut néanmoins faire intervenir un nombre important de particules dont la gestion demande des ressources calculatoires encore trop importantes pour envisager une implémentation sur un système embarqué.

Les applications de localisation par filtres Bayésiens donnent des résultats très prometteurs de l'ordre du mètre, même si elles reposent encore et toujours sur la constitution d'une base de données de mesures offline. Enfin, notons que le gain en précision n'est obtenu qu'après un certain intervalle de temps qui correspond à la durée « d'apprentissage » du filtre.

Confrontés à cette diversité de méthodes de localisation, nous avons dû sélectionner les méthodes qui nous semblaient à la fois prometteuses dans l'avenir mais aussi rapidement implémentables. Selon nous, le projet RADAR et les projets mettant en oeuvre uniquement un modèle empirique de propagation du signal avaient été supplantés par les méthodes probabilistes et ne seraient donc pas suffisants pour envisager plus tard une amélioration en précision. Telle fût notre première orientation. Or nous allons voir dans la partie suivante, qu'il sera nécessaire de confronter notre premier choix à un nouveau procédé naissant extrêmement efficace.

2. Localisation par mesure de temps

Les algorithmes utilisant le temps à des fins de géolocalisation mesurent le temps de propagation des ondes entre l'émetteur et le récepteur. Donc ces algorithmes consistent à récupérer *vitesse et le temps de transit d'un paquet entre l'émetteur et le récepteur*, malheureusement la mesure de ces deux grandeurs pose un certain nombre de problèmes de modélisation pour le premier, et de limitations liées au matériel pour le second.
Toute le mérite des algorithmes de géolocalisation qui utilisent le temps comme grandeur de référence réside donc dans l'ingéniosité qui est mise en place afin d'optimiser ces deux paramètres.
Avant de rentrer plus en détail dans les spécificités liées à ces paramètres il est nécessaire d'expliquer le protocole expérimental qui est utilisé.
La norme IEEE 802.11 fixe comme obligation au récepteur d'informer de la bonne réception des paquets en envoyant un signal de « *acknowledgement* ». Comme le montre le schéma ci-dessous, on peut utiliser une requête de *ping* (protocole ICMP) pour provoquer l'envoyer de ce signal.



La méthode fait appel au matériel suivant:

- Un point d'accès Wi-Fi (*remote node*)
- Un client composé de deux cartes Wi-Fi (une carte ne peut être à la fois active en envoyant des ping et passive en écoutant le trafic).

A partir de la trace d'écoute du trafic il est possible de récupérer les temps d'émission et de réception d'un paquet précis afin de déterminer le temps que celui-ci a mis, lors de la traversée entre le point d'accès et la carte.

Revenons maintenant aux problèmes soulevés au début de cette partie. La modélisation du temps de transit entre deux point est relativement simple dans le cas ou le paquet ne rencontre aucun obstacle. En effet, au même titre que pour la mesure de la puissance du signal il est impossible de proposer un modèle de propagation en raison de la trop grande hétérogénéité des matériaux dans les murs d'un immeuble par exemple. Voici donc ce que l'on peut lire au début d'un article de l'université technique de Berlin (TKN) au sujet de la géolocalisation à l'aide du temps:

"…The propagation time of radio signals can be used because in free air it linearly increases with the distance…"

En ce qui ce qui concerne la mesure du temps, le problème est lié aux limitations du matériel. En effet, la résolution disponible actuellement est de l'ordre de 10^{-6} secondes. Ce qui dans le cadre de la propagation du signal dans l'air (vitesse de la lumière) représente une précision d'environ 300m. On imagine bien qu'une précision de l'ordre de la centaine de mètre n'a pas grand intérêt. Pour palier ce problème, un certain nombre d'outils statiques liés ou non à la

structure du hardware sont utilisés ; nous citerons parmi eux le principe de résonance stochastique ou encore les théories liées à la structure du bruit dans un signal (bruit Gaussien…). Les meilleurs algorithmes utilisant le temps comme unité de base arrivent à obtenir une précision de 5 à 10 mètres [3].

Problèmes liés à l'implémentation de ces algorithmes dans des objets du type ubiquitaire :

Tout d'abord la modélisation du temps de propagation devient vite impossible dans des immeubles en raison des murs. Les cas d'utilisation de ce type d'algorithme sont donc assez limités.
De manière à récupérer un nombre suffisant de paquets, un grand nombre de requêtes *ping* est nécessaire, l'émetteur et le récepteur sont alors complètement saturés. Il est donc impossible dans l'état actuel de l'art d'imaginer que plusieurs clients se localisent en même temps.
Le traitement statistique et les divers filtres à mettre en place sont très gourmand en puissance de calcul et rendent ce type d'algorithme difficilement exportable sur un objet de type ubiquitaire.

C. *Problématique, justifications des hypothèses associées et grandes lignes du projet*

Le projet que nous proposons cherche à répondre à la question suivante :
« Comment élaborer un système capable de localiser un utilisateur évoluant dans une zone de l'espace couverte par un réseau de points d'accès IEEE 802.11, et de diffuser de l'information selon des régions de l'espace prédéfinies ? »

Premièrement, nous décidons de nous restreindre au milieu intérieur car le climat de Nancy en période hivernale, le manque d'équipement et de temps ne nous seront pas favorables pour effectuer des mesures récurrentes dans différentes configurations spatiales à l'extérieure. Deuxièmement, vu nos conclusions émises dans l'étude de l'état de l'art, la localisation se fera en exploitant comme grandeur physique, la puissance du signal.
Nous nous fixons comme objectif d'obtenir des résultats proches de ceux énoncés dans les articles scientifiques, selon nos propres méthodes et avec les moyens dont nous disposons. Il est important de faire la remarque suivante : étant donnés les enjeux commerciaux sous-jacents à la découverte d'un système de localisation par Wi-Fi robuste et sûr, nous avons bien

souvent été confrontés à la superficialité des articles scientifiques que nous avons lu sur le sujet. C'est pourquoi nous n'hésiterons pas à présenter dans ce rapport les phases de tâtonnement, parfois inabouties, durant lesquelles nous avons consacré beaucoup de temps à formuler les concepts extraits de nos nombreuses lectures sur le sujet et à créer des outils pour les confronter à l'expérience et analyser les résultats à la lumière de ceux déjà obtenus.

Globalement, pour donner un aperçu de la chronologie du projet, nous avons commencé par mûrement réfléchir au choix de la grandeur physique exploitable pour se repérer dans l'espace. Ce choix, comme nous l'avons déjà vu, s'est basé sur les expériences décrites dans les articles traitant du sujet et nos propres constatations expérimentales. Il nous a fallu également nous donner les moyens, aussi bien matériels que logiciels, pour effectuer des mesures de la grandeur et ensuite apprécier ses variations en fonction de paramètres tels que le temps, la distance... Nos propres études du signal ont eu une influence indéniable sur le choix de nos lectures et finalement sur le choix du modèle de localisation et de l'algorithme associé. A ce stade là, le développement de l'application sous forme d'un module graphique et d'un module implémentant le modèle, fût lancé. Il a été évidemment nécessaire de continuer les mesures expérimentales afin d'apprécier la précision de l'algorithme.

II. Etude de la grandeur physique retenue : la force du signal

Force est de constater qu'aujourd'hui il existe une grande incohérence et confusion relative à l'emploi de termes du standard 802.11. On parle de « force du signal », de « qualité du signal », de « ratio signal bruit », sans souvent connaître la définition exacte. Cette partie cherche à lever le voile sur une terminologie souvent mal employée et qui à certains moments nous a nous aussi dérouté. De cette façon nous disposerons d'un cadre linguistique précis pour discuter clairement de nos mesures.

A. *Ambiguïté de la terminologie*

1. La force du signal

La plupart du temps, les constructeurs incluent dans la suite logiciels accompagnant un quelconque matériels Wi-Fi, un client qui permet de visualiser la force du signal. Quatre « unités » sont utilisées pour représenter la force du signal radio dans la spécification 802.11 : le milliwatt (mW), le décibel référencé à un milliwatt (dBm), le RSSI (Received Signal

Strength Indicator) et une mesure par pourcentage. Elles sont plus ou moins fortement reliées entre elles mais il est toujours possible de convertir une de ces unités en l'autre.

Nous commençons par présenter les deux premières qui ne sont toutefois pas les plus couramment utilisées. La mesure en milliwatt parle plus à un physicien qui aura tendance à raisonner en termes d'énergie dissipée par unité de temps. A titre indicatif, retenons que la puissance de sortie d'un point d'accès est comprise entre 1 et 100mW tandis que celle d'un client sans fil varie plutôt entre 1 et 30 mW. L'énergie du champ électromagnétique généré par un point d'accès n'est pas commode à mesurer parce qu'elle ne s'atténue pas de façon linéaire, mais inversement au carré de la distance. La puissance atteindra donc rapidement de très petites valeurs qui ne permettront pas une interprétation facile. Le dBm n'est rien d'autre qu'une ruse mathématique pour faciliter la lecture des mesures. Souvenons-nous que lorsque la puissance en mW se voit divisée par deux, sa mesure en dBm se verra décroître d'environ 3 dBm. Nous verrons plus tard que les valeurs en dBm sont toujours négatives, preuve que la puissance environnante reste toujours en dessous du milliwatt. Pour les conversions il suffit de se reporter aux formules suivantes :

$$dBm = 10 \cdot \log(mW)$$

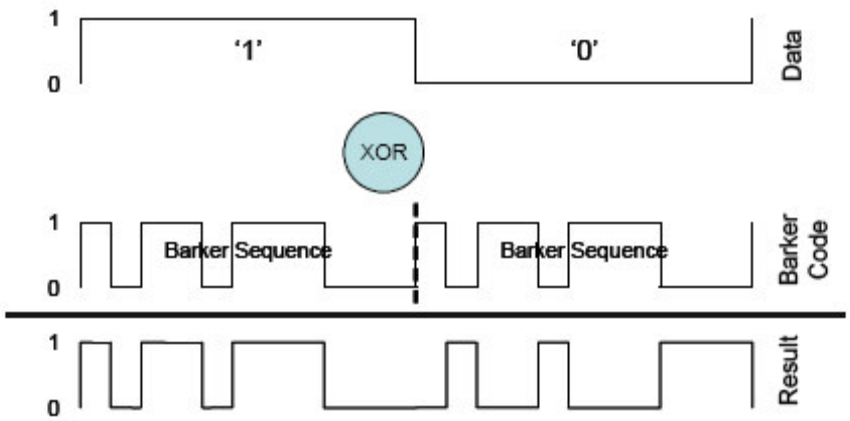
$$mW = 10^{\frac{dBm}{10}}$$

Le standard IEEE 802.11 définit un mécanisme par lequel l'énergie du champ électromagnétique doit être mesuré par l'ensemble des circuits d'une carte sans fil. Pour les standards actuels (802.11 a,b,g) la valeur numérique retournée est un entier codé sur un octet (intervalle de valeur compris en 0 et 255 en non signé) et que l'on appelle indicateur de la force du signal reçu ou RSSI. Le standard n'oblige pas les fabricants de puce a utiliser les 256 valeurs et donc chaque vendeur pourra se fixer une valeur maximale de l'indicateur RSSI (RSSI_max). Prenons par exemple Cisco qui a choisi de fixer RSSI_max à 100, ou encore Symbol qui n'utilise que les 31 premières valeurs. On comprend que le RSSI est un entier arbitraire qui assure la liaison entre le représentant du monde logiciel, à savoir le driver, et les représentants du monde physique, à savoir l'ensemble des circuits intégrés. Illustrons son intérêt par les exemples suivants. Lorsqu'une carte Wi-Fi souhaite transmettre un paquet, elle doit être capable de détecter si oui ou non le canal est libre (c'est à dire que personne d'autre n'émet à cet instant). Si le RSSI est en dessous d'un certain seuil (« Clear Channel Threshold »), la carte conclura que le canal est libre et qu'elle peut alors émettre. De même, lorsqu'une personne s'éloigne trop d'un point d'accès, il viendra un moment où le RSSI

franchira un autre seuil de réception appelé « Roaming Threshold ». Dans ce cas, la carte cherchera à s'associer à un autre point d'accès dont le RSSI mesuré sera plus élevé. Ces différents seuils sont eux aussi spécifiques à chaque constructeur dans la mesure où ces derniers ne choisissent pas dès le départ le même RSSI_max. Concernant la précision de mesure offerte par le RSSI, il est évident qu'avec une valeur de RSSI_max proche de 255, la carte atteindra une meilleure granularité. C'est un point à ne pas négliger lorsque nous sélectionnerons le matériel à utiliser dans notre système. Terminons par la mesure en pourcentage qui s'obtient tout simplement en faisant le rapport entre la valeur du RSSI mesurée et le RSSI_max. Elle a l'avantage d'autoriser la comparaison entre des mesures obtenues de différents chipsets Wi-Fi.

2. La qualité du signal

Une rapide recherche sur Internet donne accès à un foisonnement de définitions qui bien souvent semblent se substituer à la définition de la force du signal. Une telle confusion n'est pas bénigne car le terme de qualité du signal est nettement plus obscur et subtile à en croire le nombre de fois où le standard 802.11 de 1999 l'emploi dans ses spécifications (trois fois uniquement). La qualité est une notion qui se rattache à un type particulier de modulation du signal connu sous l'acronyme DSSS (*Direct Sequence Spread Spectrum*). La définition de DSSS fait intervenir un code binaire (*PN code correlation*) qui permet de vérifier ou même retraiter le message binaire reçu, de façon à ce qu'il corresponde exactement au message envoyé. Un algorithme analyse la corrélation, c'est à dire le niveau de similarité entre des paquets de bit, au moyen du code PN. Pour comprendre son fonctionnement, examinons l'exemple suivant dans lequel on cherche à transmettre la séquence de bits 10. A la transmission chaque bit d'entrée est combiné par l'opération XOR à une séquence de onze bits appelée PN-code de séquence Barker. Le schéma ci-dessus illustre l'opération.



On voit qu'avec ce système, la taille du message envoyé à travers les airs est multipliée par onze. Durant ce parcours aérien, l'onde transmise peut subir des interférences qui conduisent à des inversions de bits dans le message réceptionné. A la réception, la séquence de bit est divisée en fractions de 11 bits. Chaque fraction est ensuite comparée à la séquence « originale » codant pour un 0 ou un 1.

Symbole codant le 0 : 01001000111
Symbole codant le 1 : 10110111000
Fraction réceptionnée (1 bit inversé): 10110111001 → traduit par le bit 1

On comprend donc qu'avec un tel système de correction des erreurs il est possible d'avoir jusqu'à 5 bits inversés sans que la traduction du message original ne soit faussée. En revanche, à partir de 6 bits corrompus, des erreurs de traduction apparaîtront, conduiront à une somme de contrôle incorrecte et le paquet sera considéré comme mal formé.
Le calcul de la qualité se fonde sur une moyenne du nombre d'erreurs détectées sur une fenêtre de temps glissante.

3. Le ratio signal/bruit (Signal Noise Ratio)

Le ratio signal/bruit est habituellement compris par les administrateurs de réseau 802.11 comme le rapport entre la puissance du signal de données et la puissance naturellement dégagée dans le milieu ambiant ou bruit. Il faut tout de même nuancer cette définition qui dépend évidemment du type de bruit retenu. En effet, si l'on adopte le point de vue d'un ingénieur en électronique, le SNR sera vu comme le rapport entre la force du signal de données et le bruit intérieur de la puce électronique réceptrice. Les circuits électroniques sont en effet tous sujets à un type de bruit connu sous le nom de bruit de Boltzmann qui tire ses origines d'effets thermiques. La chaleur dégagée par effet joule au sein des puces entraîne un certain nombre de distorsions électriques qui se manifestent elles-mêmes sous forme de bruit. Les circuits électroniques peuvent aussi être perturbés par du bruit provenant de sources externes ; par exemple, aux Etats-Unis il est courant de mesurer un bruit de fréquence proche des 60 Hz (fréquence à laquelle le courant alternatif américain est transporté). Concluons donc sur le fait que nous qualifierons d'« interférence » l'énergie radio-fréquence ambiante. Nous laisserons ainsi le terme de ratio signal/bruit à l'ingénieur en électronique, et parlerons plutôt de ratio signal/interférence dans le cadre des réseaux 802.11.

B. *Hardware et Software*

Comme toutes les technologies à la mode, le Wi-Fi au même titre que son cousin le Bluetooth se retrouve sous de multiples formes d'antennes, cartes PCMCIA etc.
Dans cette partie, après une brève introduction de la norme IEEE 802.11, nous présenterons les différences de possibilités offertes entre Windows et Linux. Ensuite, nous nous attarderons sur un coté plus hardware et plus technique en faisant un panorama des différents types de puces et des drivers qui permettent de les utiliser sous Linux.

1. La norme 802 11

Créée en 1997 par l'IEEE la norme 802 11 autorise à l'origine un transfert de données de l'ordre de 2Mbps sur la bande 2,4GHz ouverte à l'utilisation publique sans licence. En 1999, l'IEEE adopte deux nouvelles extensions à cette norme la 802.11a et la 802.11b. La 802 11a fonctionne sur la plage 5Ghz à un débit théorique de 54Mbps. Cette norme exploite 12 canaux qui ne se chevauchent pas, et malgré le débit plus élevé cette norme est nettement moins utilisée que la 802.11b pour plusieurs raisons. En effet, la fréquence supérieure à la norme 802.11b diminue la portée, augmente la puissance à fournir alors que la puissance des produits embarqués est limitée et enfin nécessite une autorisation pour l'utilisation en extérieur (fréquence de l'armée). Malgré ces défauts, cette norme est de plus en plus utilisée et, dans le cadre de la géolocalisation, sa faible utilisation évitait les problèmes d'interférences La norme 802.11b également appelée *High Rate DS*, est une extension de la modulation de l'étalement du spectre à séquence directe (DSSS – *Direct Sequence Spread Spectrum*). Elle utilise un spectre de 14 canaux qui se chevauchent. Elle fut dès 1999 plébiscitée par les constructeurs de la Wi-Fi Alliance (groupe composé de la quasi intégralité des industriels du secteur) ce qui explique sa notoriété aujourd'hui. Naturellement les produits utilisant cette norme sont les moins chers.

2. Mode opératoires des clients 802 11

Il existe deux principaux modes opératoires pour les clients de la famille 802.11, le mode *ad-hoc* et le mode *infrastructure*, les deux autres modes, *master* et *monitor* sont beaucoup moins utilisés mais sont indispensables au développement d'application de géolocalisation. Nous détaillerons particulièrement le mode monitor car il est justement celui

utilisé dans le cadre de la géolocalisation mais aussi pour d'autres activités comme le *Wardriving.*
C'est le mode dans lequel se met la carte lors de l'utilisation du célèbre logiciel *Kismet* dont nous nous sommes inspirés pour réaliser le sniffer qui sera plus détaillé par la suite.

Le mode *ad-hoc* est conçu spécialement pour les communications poste à poste ; pour communiquer dans ce mode il faut que les deux postes utilisent le même SSID. Les clients ad-hoc ne s'annoncent pas eux-même. Cependant, contrairement au mode infrastructure ce mode ne possède pas d'algorithme de contrôle de l'émission. En effet, un client peut décider d'émettre sans garantie aucune que le paquet sera envoyé dans des conditions qui assurent sa réception. De plus, ce mode est limité à la communication deux à deux.

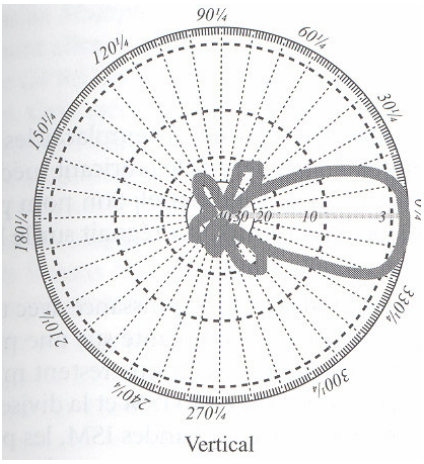
Le mode *infrastructure* nécessite un point d'accès (en mode *master*) qui diffuse son SSID désignant le nom du réseau. Les algorithmes de géolocalisation nécessitent tous des points d'accès, qui diffusent ou nom le nom du réseau. Ce qui est important c'est l'adresse MAC de la carte de l'antenne qui permettra de l'identifier.

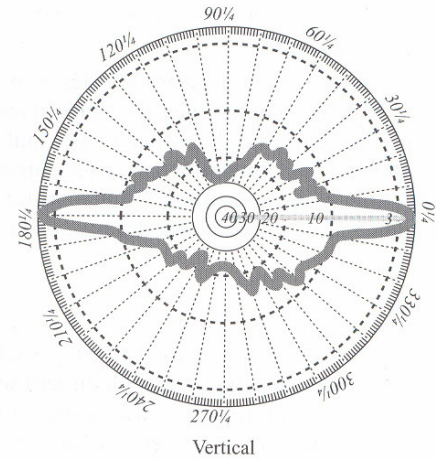
Le mode *monitor* convertit la carte en récepteur passif totalement invisible pour le réseau. Dans ce mode la carte est uniquement capable de recevoir des paquets, il faut noter que toutes les cartes Wi-Fi ne permettent pas de passer dans ce mode souvent à cause des pilotes.

Le mode *master* convertit la carte en un point d'accès qui est en général géré par un micrologiciel spécifique de manière à optimiser le trafic.

3. Les antennes

Sans passer par une présentation de la physique d'une antenne qui dépasse largement le cadre de notre étude, il est intéressant d'avoir un panorama sur les divers types d'antennes de manière à comprendre les problèmes de diffusion du signal.





Nous pouvons voir sur les abaques ci- les différences qui existent dans la propagation des signaux. On peut noter les deux principaux types d'antennes : directionnelle et omnidirectionnelle

Notons l'arrivé récemment des antennes MIMO qui possèdent des algorithmes identiques à ceux présents dans les antennes GSM, qui sont capables de jouer sur l'ouverture du faisceau de manière à rendre l'antenne plus ou moins directionnelle et à atteindre des clients en limite de portée. Ce genre d'antennes pose des problèmes dans le cadre d'algorithmes de géolocalisation utilisant des mesures préalables pour définir la position d'un client. En effet la puissance à un point donné pouvant varier très fortement il devient impossible de se fier à la puissance du signal.

4. Windows VS Linux

Dans les premier temps du projet nous avons utilisé Windows afin de sniffer les réseaux sans fil. De cette manière nous avons pu nous faire rapidement une idée sur les variations des différentes grandeurs comme la puissance du signal, le bruit etc. Malheureusement, le logiciel que nous utilisions à ces fins, le très connu *NetStumbler* semblait ne pas être capable de nous fournir plus de 2 mesures par secondes alors que nous avions paramétré le point d'accès sur l'envoi de 10 paquets *beacon* à la seconde (préciser ce qu'est une balise). De plus, le bruit du signal était toujours le même et ce en utilisant des cartes issues de divers constructeurs. Nous nous sommes donc rapidement heurtés au problème de la gestion du Wi-Fi sous Windows. En effet, contrairement à Linux où l'on peut accéder directement à la socket sur laquelle écrit la carte, Windows à travers sa couche *d'adapter* Wi-Fi limite l'accès à ces informations. Cependant la configuration de Linux pour

le sans fil n'est pas non plus une chose facile. Car si tous les fournisseurs ont des drivers pour Windows il n'en est pas forcément de même pour Linux. Pour certains constructeur de cartes ils faut compter sur l'ingéniosité et la capacité de partage de certains hackers sur leurs travaux de *reverse engineering*. On peut noter aussi que certains projets comme Ndiswrapper (http://ndiswrapper.sourceforge.net/) permettent d'utiliser les drivers Windows sous Linux. Même si cela semble marcher, nous ne l'avons pas essayé dans le cadre de notre projet.
Dans le point suivant nous aborderons les diverses puces qui existent, les divers types de drivers et enfin comment les installer sous Linux. Mais avant tout cela, une brève synthèse de la situation s'impose :

- Windows ne permet pas d'utiliser au maximum le matériel Wi-Fi mais il permet au moins d'accéder rapidement et facilement, à des données de manière à se faire une idée.
- Linux permet de vraiment accéder au plus près des informations de la carte si tant est que les drivers soient disponibles et que l'on arrive à les installer.

Dans le cadre de notre application qui a pour objectif d'être intégré sur une plateforme de type ubiquitaire, la maîtrise du Wi-Fi sous Linux semble obligatoire.

La mise en place d'un environnement Linux adéquat pour l'utilisation de notre carte PCMCIA Netgear WPN511FS a demandé de notre part une très grande patience. Notre objectif était d'installer correctement les drivers Madwifi pour faire fonctionner la carte. Dans la grande majorité des cas, ce driver n'est pas intégré par défaut aux grandes distributions Linux (SuSe, Mandrake...). Ceci étant, nous avons donc dû apprendre à nous servir du célèbre utilitaire *Make* pour compiler les sources du driver. Mais la difficulté n'était pas pour autant contournée car pour compiler Madwifi il nous a fallu bien souvent recompiler le noyau Linux. Désormais nous sommes capables de compiler sans problèmes un noyau linux et de le rendre bootable au démarrage de la machine. A ce niveau, nous étions capables de compiler Madwifi et de l'installer. Néanmoins d'autres problèmes ont vu le jour ; principalement des problèmes d'incompatibilités de versions entre les drivers Madwifi, la librairie Libpcap, que nous verrons plus tard, et le sniffer Ethereal. Souvent ces problèmes nous ont conduits à mesurer des valeurs de force du signal aberrantes. Afin de résoudre cela, nous avons pris la décision de construire notre environnement Linux personnalisé et minimal pour éviter tout conflit. Le challenge fût pris de mettre en place une distribution Gentoo. La particularité de la distribution Gentoo est qu'elle ne dispose pas de logiciel d'installation et qu'il faut donc la

construire « à la main ». De plus, elle ne fournit que très rarement des paquetages précompilés (le serveur X11 de X.org et KDE l'étaient heureusement) et il faut donc généralement faire preuve de patience pour compiler les logiciels. Malgré l'effort demandé, nous ne changerions plus pour une distribution alternative car l'environnement Wi-Fi installé fût d'une stabilité irréprochable.

5. Les puces Wi-Fi

Bien que les fournisseurs de matériels Wi-Fi soient nombreux, les puces radio proviennent du même nombre restreint de fournisseurs. Parmi les plus importants on citera Prism2/Intercil, Cisco/Aironet, Wavelan/Orinoco et Atheros, tous les quatre issus de nombreuse fusions, scissions, ou encore acquisitions.

Prism2 a licencié son schéma de puce en 2000 et depuis beaucoup de constructeurs l'ont adopté ce qui en fait la puce la plus utilisée. Les drivers Prism2 (et maintenant Prism54) sont donc compatibles avec un grand nombre de cartes. Toutefois, deux drivers compatibles pour une même carte ne donnent pas accès aux mêmes fonctionnalités : par exemple, le pilote *HostAP* pour carte Prism2 gère le mode point d'accès (master) alors que le pilote *wlan-ng* ne le fait pas.

Atheros quant à lui est le leader des cartes à base de 802.11a et se distingue en utilisant sa propre puce qui ne tient pas compte du standard Prism2. Pour satisfaire aux contraintes du FCC et sécurisé le paramétrage de la puissance d'émission, Atheros a développé une couche d'abstraction HAL (*Hardware Abtraction Layer*) qui s'intercale entre le matériel et son pilote. Parmi les drivers qui s'appuient sur HAL on trouve le célèbre pilote Madwifi crée en collaboration avec Atheros.

Pour des informations plus exhaustives le site de Jean Tourrilhes http://www.hpl.hp.com/personal/Jean_Tourrilhes/ créateur des *Wireless Tools* de Linux et ingénieurs chez HP est une référence. Pour un suivi plus pratique des nouveaux drivers et matériels le site http://www.lea-linux.org/cached/index/Driver:Wifi.html fournit toutes les informations. Il faut noter que la majorité des distributions grand public dont *Debian*, *Red Hat* et *Mandriva* gèrent aujourd'hui automatiquement les périphériques sans fils.

Pour installer une carte Wi-Fi sous Linux deux éléments sont nécessaires :

- Le pilote correspondant à la carte
- Le logiciel *Wireless Tools*

A première vu rien de très compliqué mais cependant connaître la puce qui est dans sa carte n'est pas forcément une chose facile. En effet, les constructeurs de matériels ne disent que très rarement sur quelle puce s'appuie leur carte. Dans ce cas il faut se résoudre à utiliser des techniques d'identification comme la commande PCMCIA cardctl ident qui renvoie la chaîne d'identification des matériels connectés aux ports PCMCIA (ou lspci –v pour les cartes pci).
Les Wireless Tools fournissent toute une panoplie d'outils qui permettent de tirer parti des capacités de la carte. En fonction de la carte et du driver, toutes les fonctionnalités ne sont pas disponibles ou sont moins bien gérées. Voici les quatre principales commandes de ce logiciel :
iwconfig : Permet de configurer la carte, son mode son SSID, la clé WEP etc.
iwlist : Permet principalement d'afficher la liste de commandes que supporte la carte. Il dispose aussi d'une fonctionnalité qui permet d'afficher la version de Wireless Tools installée.
iwspy : Affiche la qualité de la connexion avec un ou plusieurs réseaux (nœuds) différents.
iwpriv : Permet de configurer des options sans fils privées, autrement dit des paramètres qui ne sont applicable que à cette carte. Il se différencie de *iwconfig* qui lui ne gère que les paramètres standards.
Dans notre projet le code source de ces quatre commandes ainsi que des sources du logiciel *Kismet* nous ont inspiré dans le développement de notre propre sniffer.

C. *Propriétés de la force du signal*

1. Variations temporelles

Les expériences décrites ci-dessous avaient pour objectif de mettre en évidence une variabilité temporelle suffisamment faible en position fixe. Pour que la grandeur soit exploitable nous devions apporter la preuve d'un couplage faible entre la puissance mesurée et l'heure de mesure.

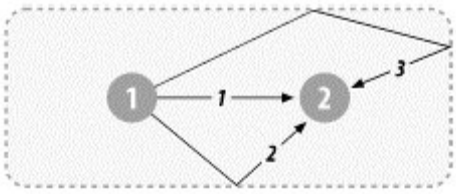
Expérience 1 :
Elle a consisté à nous positionner à 1m d'un point d'accès de marque Belkin et à enregistrer les valeurs de puissance du signal durant 10 minutes.

Analyse 1 :

Les courbes révèlent que les valeurs du signal se répartissent sur une bande large d'environ 10 dBm. En calculant les fréquences d'apparition de chaque valeur, nous observons qu'il existe deux à trois valeurs prédominantes dans le « spectre » des valeurs. Ce sont elles qui auront l'influence maximale sur la moyenne de la puissance. Il peut être intéressant de voir l'évolution de ces proportions sur une échelle de temps d'abord très courte (toutes les 10 secondes), puis plus longue (toutes les 2 minutes). Nous constatons que le calcul des proportions à petite et grande échelle montre que, selon le cas, la proportion maximale peut jongler entre les deux à trois valeurs prédominantes du spectre. Ces légères variations sont liées à un ensemble de perturbations environnementales.

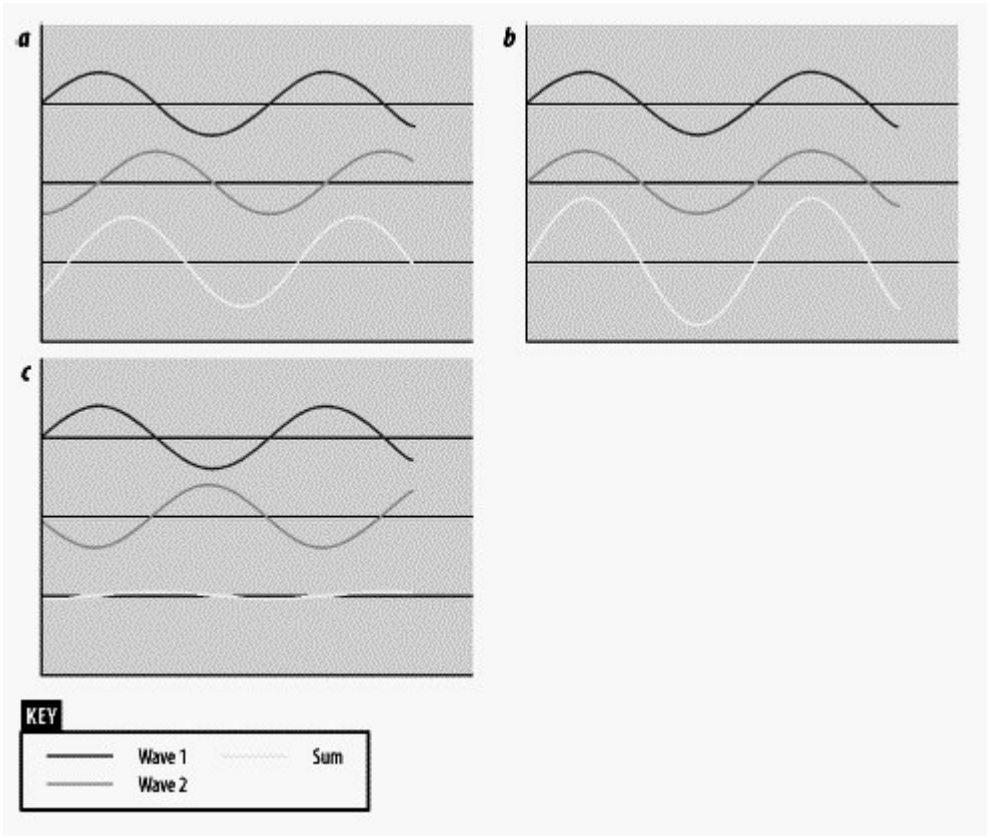
Des expériences sur des intervalles plus longs ont mis en évidence qu'en période de nuit, lorsque le bâtiment était vide, les variations devenaient extrêmement faibles. Souvenons-nous que les systèmes Wi-Fi utilisent une gamme de fréquence étroite comprise entre 2,4 à 2,4835 GHz. Celle-ci correspond à la gamme des micro-ondes. Or nous savons que les molécules d'eau soumises à un rayonnement de fréquence avoisinant les 2,45 GHz se mettent à vibrer. L'onde de son côté va elle-même subir l'influence de ces vibrations sous forme de perturbations. Comme le corps humain est constitué à 80% d'eau, nous comprenons maintenant pourquoi le déplacement de personnes dans le bâtiment peut venir augmenter les perturbations du signal.

Un phénomène moins évident mais tout aussi présent est celui des interférences liées aux chemins multiples empruntés par une même onde. Il apparaît plus facilement à l'utilisation d'antennes omni-directionnelles (i.e dispersion égale de l'énergie radio-fréquence sur 360°). Les ondes sont diffusées dans toutes les directions et peuvent alors être réfléchies à plusieurs reprises par les surfaces (murs, sol, plafond ...).



L'illustration sert d'exemple simplifié dans le cas d'un point d'accès émettant en direction d'une carte réceptrice. A l'arrivée les trois ondes issues pourtant de la même transmission, vont venir s'ajouter les unes aux autres de sorte que, dans le pire des cas, l'onde résultante prenne la forme de l'onde nulle. Le récepteur ne comprendra donc pas la transmission car à

ses yeux il n'y aura jamais eu de signal transmis. La figure ci dessous illustre ce principe de combinaison d'onde par superposition.



Généralement, une modification de l'orientation du récepteur suffit à pallier le problème. Nous aurons d'ailleurs l'occasion de discuter un peu sur l'influence de l'orientation dans les mesures de puissance.

Nous pensons que la constitution d'une carte des puissances est un moyen, certes grossier, de nous affranchir d'une partie des perturbations sur le signal. En outre, il semble indispensable de toujours travailler sur un lot de mesures et non pas sur une mesure unique des signaux afin d'extraire la valeur centrale qui nous intéresse.

Expérience 2 :

Dans deux des articles que nous avons lus [3,5,8], les chercheurs font usage de la fonction d'auto-corrélation pour détecter si les valeurs des échantillons successifs mesurés sont réparties de façon hasardeuse ou si au contraire elles sont corrélées sur un certain intervalle de temps. Si l'échantillon est hasardeux, la fonction d'auto-corrélation est proche de 0. Dans le cas contraire, il est possible de déterminer sur quel intervalle de temps régulier les valeurs sont le plus liées entre elles. La fonction d'auto-corrélation a été calculée sur plusieurs de nos

captures, en des temps et lieux différents et pour quatre points d'accès différents (2 modèles Netgear différents, 1 modèle D-Link, et 1 modèle Belkin). Nous livrons ci-après les aspects théoriques de son calcul et l'analyse des résultats observés.

Analyse 2 :

En supposant un ensemble d'échantillons $Y_1, Y_2 ... Y_N$ obtenus à des temps $T_1, T_2 ... T_N$ on définit :

Le coefficient R_h d'autocorrélation: $R_h = \frac{C_h}{C_0}$

Avec C_h l'autocovariance : $C_h = \frac{1}{N} \cdot \sum_{t=1}^{N-h} \left(Y_t - \overline{Y}\right) \cdot \left(Y_{t+h} - \overline{Y}\right)$

Et C_0 étant la variance de l'ensemble de mesure: $C_0 = \frac{1}{N} \cdot \sum_{t=1}^{N} \left(Y_t - \overline{Y}\right)^2$

De cette façon on analyse si les valeurs Y_t et Y_{t+h} sont corrélées entre elles sur la durée $X_{t+h} - X_t$ qui les séparent.

Il apparaît que pour les points d'accès de la marque Netgear, un effet de corrélation notoire existe entre des mesures espacées de $(14 * BeaconInterval)$ secondes. Cette valeur que nous avons trouvée empiriquement, ne dépendait pas du temps ni du lieu où se faisait le scan. De la même façon, le point d'accès D-Link émettait des beacons dont la puissance reçue était fortement corrélée mais selon cette fois un autre intervalle de temps $(9 * BeaconInterval)$, lui aussi indépendant de la date et du lieu de la mesure. Enfin, il n'a été observé aucun phénomène d'auto-corrélation entre les mesures faites sur le point d'accès Belkin.

Nous pensons que l'auteur d'un système de localisation par Wi-Fi baptisé Horus [5,6,7,8] a effectivement détecté un phénomène d'auto-corrélation au niveau des mesures du signal. Cela s'explique peut-être par des phénomènes physiques liés à la vibration des cristaux des émetteurs et récepteurs Wi-Fi que nous ne connaissons pas. Néanmoins, nous doutons que l'auto-corrélation des valeurs soit en permanence observable pour tout type de points d'accès comme a pourtant l'air de le suggérer le concepteur d'Horus.

2. Variations spatiales

Les mesures qui vont suivre nous ont permis d'écarter définitivement l'éventualité d'un système de localisation fondé sur une modélisation théorique de la force du signal dans un espace intérieur.

Expérience 3 :

Dès le début nous avons focalisé notre attention sur le tracé des variations du signal en fonction de la distance qui nous éloignait du point d'accès. Nous analysons ici trois lots de mesures. Le premier a été effectué en milieu intérieur avec un point d'accès de type D-Link dans le couloir du 4^{ième} étage de l'Ecole des Mines, entre 1m et 8m. Les second et troisième lots correspondent à des mesures effectuées à l'extérieur avec un point d'accès Belkin. L'un des scans couvre l'intervalle de 2m à 32m par pas de 2m, l'autre couvre l'intervalle de 4m à 38m par pas de 4m. Quelque soit le lot en question, la capture du signal se faisait pendant une minute.

Analyse 3 :

Première constatation à la vue des courbes de moyenne, la puissance du signal a tendance à décroître en fonction de la distance. Notons que c'est sur les premiers mètres que la variation est la plus forte. A mesure que l'on s'éloigne du point d'accès, la variation des moyennes se fait moindre et il se peut alors que la théorique décroissance de la courbe ne soit pas respectée. En milieu intérieur ce phénomène s'observe dès 5m tandis qu'à l'extérieur il n'apparaît qu'au bout de 20m. Dans une configuration unidimensionnelle nous pouvons donc affirmer que la variation de sensibilité du système dépendra des variations de la dérivée de la courbe de puissance. Les graphiques sous forme de nuages de points donnent un aperçu de l'étalement temporel des valeurs des lots et des recouvrements éventuels entre des lots constitués en des positions différentes. Même en ayant recourt au principe de moyennes mobiles sur une fenêtre de dix mesures pour lisser les variations mineures, force est toutefois de constater qu'il y a régulièrement des moments où nous mesurons une même valeur de signal pour des éloignements différents.

Expérience 4 :

Après avoir étudié les variations de la force du signal à moyenne échelle, il nous a semblé judicieux d'effectuer des mesures pour observer les variations à petite échelle. L'idée était de nous rapprocher le plus possible de la longueur d'onde d'un signal à 2,45 GHz (soit environ 12 cm) pour éventuellement déceler des propriétés intéressante. L'expérience a donc consisté à faire des mesures tous les 10cm, pour des distances comprises entre 20cm et 1m d'un point d'accès Belkin.

Analyse 4 :

La courbe des moyennes met en évidence qu'une variation de l'ordre de 7,5 dBm sur une distance d'à peine 10cm est tout à fait possible. En admettant que notre système de localisation soit embarqué et manipulé par un utilisateur humain, on affirme clairement que celui-ci ne restera jamais complètement immobile (influence de la respiration, léger mouvement) et que ceci viendra perturber la mesure du signal. Il faut admettre que contrer les variations de très faible échelle s'apparente plus à un défi qu'à une partie de plaisir. Quoiqu'il en soit, nous n'envisageons pas pour le moment d'atteindre des précisions de l'ordre de la dizaine de centimètres, et l'élimination des perturbations de faible échelle se fera tout naturellement, lors la construction de la carte des signaux, par le choix d'un pas entre points de mesures suffisamment grand par rapport à la longueur d'onde.

Expérience 5 :

Très tôt le projet RADAR [1] a mentionné le problème de l'orientation de l'antenne réceptrice. Pour nous en convaincre nous avons à partir d'une position fixe, orienté notre carte Wi-Fi selon quatre directions principales afin d'observer d'éventuelles variations du signal.

Analyse 5 :

A l'observation des courbes obtenues, nous concluons qu'il existe a priori un delta de moins de 5 dBm entre chacun des relevés pour une orientation donnée. Toutefois rien ne nous permet de trancher clairement entre influence ou non de l'orientation car, quelle que soit l'expérience réalisée, nous restons toujours dépendants des perturbations environnementales. Le fait que les courbes se chevauchent et ne sont donc pas suffisamment « séparées », nous a conduit à oublier temporairement le léger phénomène de variation dû à l'orientation.

III. Méthode adoptée pour le calcul de position

Cette partie s'attachera tout d'abord à décrire les concepts fondamentaux relatifs au filtre Bayésien que nous avons choisi d'implémenter. Nous décrirons ensuite l'algorithme mis en oeuvre avec le modèle mathématique qui lui est associé.

Globalement le principe du système peut se résumer ainsi : à partir de mesures expérimentales effectuées en de multiples points de l'espace, nous construisons une carte des probabilités a priori. En mode localisation et sur la base de la précédente carte qui constitue en quelque sorte la « connaissance » de notre filtre Bayésien, nous mesurerons en temps réel une empreinte du signal qui, confrontée à la carte de probabilité, met à jour une carte de probabilité de présence.

A. *Préliminaires sur les filtres Bayésiens*

L'objectif du filtrage Bayésien est de fournir une distribution de probabilité de présence de l'utilisateur sur l'ensemble des positions de la carte qui lui sont accessibles. Grâce à cette représentation il devient possible d'apprécier la vraisemblance de chaque position. Si le filtre que nous choisirons s'appuiera sur une discrétisation de l'espace sous la forme d'une grille, nous choisissons néanmoins d'aborder le cadre théorique de la manière la plus générale qui soit, c'est à dire d'un point de vue continu.

Comme son nom l'indique, le filtrage Bayésien s'appuie sur la célèbre formule de Bayes :

$$P(X/Y) = \frac{P(Y/X) \cdot P(X)}{P(Y)}$$

Supposons que la variable aléatoire X décrive la position de l'utilisateur et que la variable Y décrive une perception de l'environnement (une mesure de la puissance du signal par exemple). Grâce à la formule de Bayes il est donc possible d'estimer la probabilité de position de l'individu $P(X/Y)$ connaissant une perception. Pour effectuer le calcul, il convient tout d'abord de disposer de $P(Y/X)$ qui correspond à la probabilité d'observer une perception connaissant la position. Elle s'obtient justement en construisant une carte de l'environnement à partir du modèle de capteur choisi ; notre carte *offline* du signal servira à cet effet. Ensuite, $P(X)$ est la probabilité de présence pour chaque position de la carte. Nous poserons à l'initialisation du filtre que la probabilité sera répartie uniformément sur l'ensemble de la carte et qu'ensuite, de façon récursive, elle sera égale à l'estimation précédente de la

probabilité de positions. Enfin pour estimer $P(Y)$, nous rusons formellement à l'aide de la loi des probabilités marginales qui est la suivante :

$$P(Y) = \sum_X P(Y / X) \cdot P(X)$$

Ainsi le calcul de $P(X / Y)$ par la loi de Bayes pourra s'écrire généralement sous cette forme :

$$P(X / Y) = \eta \cdot P(Y / X) \cdot P(X)$$

$$\eta = \sum_X P(Y / X) \cdot P(X) \text{ , } \eta \text{ jouant le rôle de normalisation}$$

Nous venons donc de voir comment il était possible de mettre à jour une estimation de probabilité de position à partir de l'acquisition d'une perception environnementale. Or nous aimerions bien aussi doter notre modèle d'un moyen de *tracking* pour intégrer un déplacement éventuel de notre individu qui a priori ne restera pas statique. Pour cela, il suffit de reprendre la loi des probabilités marginales écrite sous la forme suivante :

$$P(X / D) = \sum_{X'} P(X /D, X') \cdot P(X')$$

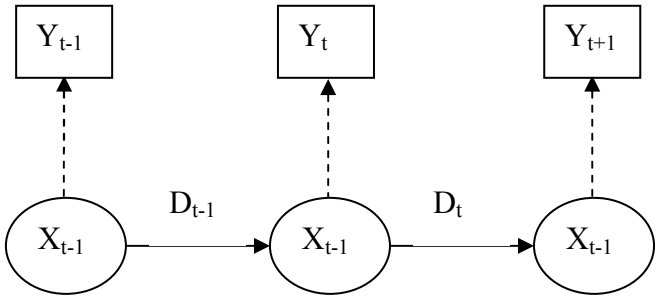
Le terme $P(X / D, X')$ correspond à la probabilité que l'individu positionné en X' et exécutant l'action D se retrouve en X. Ce modèle de déplacement sera ajusté selon le comportement habituel de l'utilisateur. Comme avant $P(X')$ est issu d'une estimation de position précédente.

Généralement à un temps t et étant en position x_t, nous disposerons d'une série d'informations obtenues aux temps précédents relatives à la fois aux décisions de déplacement d_t et aux perceptions environnementales y_t. Cependant pour calculer récursivement $P(x_t / d_1, y_2, ..., y_{t-1}, d_{t-1})$ il nous faut faire des hypothèses sur la chaîne de Markov qui nous sert de modèle : nous supposerons d'une part que les perceptions ne dépendent que de l'état courant x_t , et que d'autre part la position x_t acquise suite à un déplacement d_{t-1} ne dépend que de la position précédente x_{t-1} . Nous traduisons respectivement ces deux hypothèses par les égalités qui suivent :

$$P(y_t / d_1, y_2, ..., d_{t-1}, y_{t-1}, x_t) = P(y_t / x_t)$$

$$P(x_t / d_1, y_2, ..., y_{t-1}, d_{t-1}, x_{t-1}) = P(x_t / d_{t-1}, x_{t-1})$$

Le schéma ci-dessous explicite la chaîne de Markov sur laquelle nous travaillons :



Récapitulons sous forme de liste, les éléments nécessaires à la constitution d'un filtre Bayésien :

- Une carte du signal *offline* construite à partir de mesures expérimentales et d'un modèle de capteur ; elle donne accès à la probabilité $P(y / x)$
- Une répartition initiale de la probabilité de présence de l'utilisateur $P_0(x)$; n'ayant aucune raison de privilégier une position plus qu'une autre à l'instant t_0, nous décidons arbitrairement de répartir la probabilité de manière uniforme.
- Un modèle décrivant les déplacements possibles de l'individu et donnant alors la probabilité $P(x / x', u)$
- Une série d'informations sur le déplacement et la mesure du signal $d_1, y_2, ..., d_{t-1}, y_t$

Lorsqu'on décrit un filtre Bayésien, il est d'usage de déclarer la fonction *Belief State* déjà vu plus haut de la façon suivante :

$$Bel(x_t) = P(x_t / d_1, y_2, ..., d_{t-1}, y_t)$$

Nous allons à présent transformer cette égalité de manière à trouver une relation récursive donnant à partir de $Bel(x_{t-1})$, du déplacement d_{t-1} et de la perception y_t, la valeur de $Bel(x_t)$.

Définition de *Bel*:

$$Bel(x_t) = P(x_t / d_1, y_2, ..., d_{t-1}, y_t)$$

Appliquons la loi de Bayes :

$$Bel(x_t) = \eta \cdot P(y_t / d_1, y_2, ..., d_{t-1}, x_t) \cdot P(x_t / d_1, y_2, ..., d_{t-1})$$

Où : $\eta = \sum_{x_t} P(y_t / d_1, y_2, ..., d_{t-1}, x_t) \cdot P(x_t / d_1, y_2, ..., d_{t-1})$

Appliquons l'hypothèse de Markov 1 :

$$Bel(x_t) = \eta \cdot P(y_t / x_t) \cdot P(x_t / d_1, y_2, ..., d_{t-1})$$

Appliquons le théorème des probabilités totales :

$$Bel(x_t) = \eta \cdot P(y_t / x_t) \cdot \int P(x_t / d_1, y_2, ..., d_{t-1}, x_{t-1}) \cdot P(x_{t-1} / d_1, y_2, ..., d_{t-2}, y_{t-1}, d_{t-1}) \cdot dx_{t-1}$$

Appliquons l'hypothèse de Markov 2 :

$$Bel(x_t) = \eta \cdot P(y_t / x_t) \cdot \int P(x_t / d_{t-1}, x_{t-1}) \cdot P(x_{t-1} / d_1, y_2, ..., d_{t-2}, y_{t-1}) \cdot dx_{t-1}$$

Appliquons la définition de *Bel*:

$$\boxed{Bel(x_t) = \eta \cdot P(y_t / x_t) \cdot \int P(x_t / d_{t-1}, x_{t-1}) \cdot Bel(x_{t-1}) \cdot dx_{t-1}}$$

La base théorique des filtres Bayésiens est contenue essentiellement dans l'égalité ci-dessus. Nous allons voir comment l'adapter au cas discret et comment l'intégrer sous la forme d'un algorithme. Pour l'instant nous la gardons en mémoire le temps de construire le cadre mathématique permettant de remplir les hypothèses nécessaire à l'utilisation de la formule.

B. *Cadre mathématique*

Le système de localisation que nous construisons se restreindra pour le moment à considérer l'espace physique sous la forme d'un espace à deux dimensions désigné par E. Nous discrétisons cet espace en définissant un ensemble fini composé de vecteurs de E et noté $X = \{x_1, ..x_i .., x_p\}$. Dans notre façon de penser, nous imaginons un découpage de la carte sous la forme d'une grille régulière dont chaque centre de cellule correspond à un vecteur de

position appartenant à X. La gamme de valeurs de puissance du signal utilisée par le driver Madwifi constitue un ensemble fini $V = \{v_1,...,v_m\}$. Nous supposerons qu'à chaque position x_p il est possible de capter une empreintes de N signaux provenant des N points d'accès composant notre réseau IEEE 802.11. Dès lors, étant en position x_i, nous pourrons capter à chaque instant une empreinte de la forme $y_i = (y_{1,i},..y_{k,i}..,y_{N,i})$ où les $y_{k,i}$ prennent leur valeur dans V. Lors de la création de la carte *offline* du signal, nous calculons pour chaque position x_i la moyenne des empreintes reçues pendant un intervalle de temps donné (une minute dans notre cas) notée $\mu_i = (\mu_{1,i},..\mu_{k,i}..,\mu_{N,i})$.

La phase *online* consiste ensuite à confronter une empreinte $y_t = (y_1...y_N)$, relevée à un temps t et à partir d'une position inconnue, à celles enregistrée durant la phase *offline*.

Le filtre Bayésien nécessite, on le rappelle, le calcul de $P(y_t / x_i)$. Pour modéliser facilement la puissance du signal, nous avons choisi d'utiliser une loi normale car elle s'adapte convenablement aux histogrammes de proportions expérimentaux. Ainsi, si pour un point d'accès k et une position x_i donnés la moyenne du signal *offline* valait $\mu_{k,i}$ alors on estimera $P(y_t / x_i)$ par :

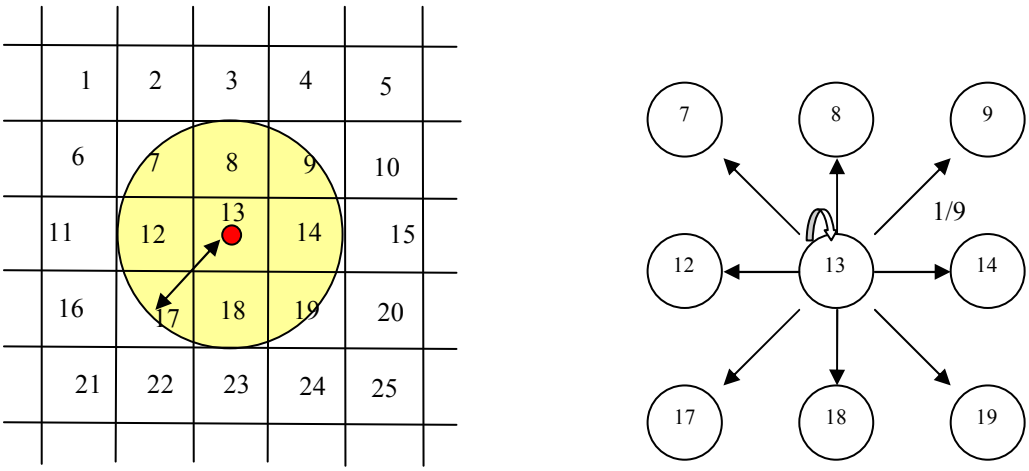
$$P(y_t / x_i) = P(y_1 \cap ... \cap y_N / x_i) = \prod_{k=1}^{N} G_{k,i}(y_k)$$

$$G_{k,i}(s) = \frac{1}{\sigma \cdot \sqrt{2 \cdot \pi}} \cdot \int_{s-0.5}^{s+0.5} e^{-\frac{(x-\mu_{k,i})^2}{2 \cdot \sigma^2}} dx$$

La première égalité fait apparaître un produit justifié par l'hypothèse d'indépendance entre les évènements associés à chaque mesure de signal par point d'accès.
La seconde égalité n'est autre que le calcul de la probabilité $P(s-0.5 \le s \le s+0.5)$ selon la loi normale. La moyenne est égale à celle calculée lors de la phase *offline* et la variance sera fixée arbitrairement par nos soins. Ce choix de modéliser le signal par une Gaussienne a découlé des histogrammes que nous avons construits lors de l'analyse du signal, et optimise le

stockage en mémoire de la carte car il nous suffit de ne conserver, par capture et par signal, que deux nombres : la moyenne et éventuellement la variance.

Il nous reste à définir le modèle de déplacement de l'individu car nous souhaiterions être capables de réaliser un suivi dynamique. Nous supposons que la personne se déplace à une vitesse de croisière de 5 km.h^{-1} soit environ 1.39 m.s^{-1}. Etant donnée une cellule de centre x_i et une grille de pas λ, on imagine un disque centré sur x_i de rayon 1.39 m.s^{-1}. A partir de là, nous pouvons déterminer les cellules environnantes que l'individu pourrait rejoindre en partant de x_i après une seconde. Nous construisons alors un graphe à nombre d'états fini dont les arcs correspondent à la possibilité de naviguer d'un état à un autre après une seconde. Chaque arc orienté sortant d'un état donné porte une probabilité calculée uniformément d'après le nombre total d'arc sortant de l'état en question. La structure de donnée choisie pour intégrer le modèle de déplacement est une matrice carrée $P \times P$ avec P le nombre de cellule composant la carte. L'illustration ci-dessous explicite le principe de calcul.



Notons qu'il serait plus précis d'attribuer entre deux états une probabilité qui soit fonction de l'aire du disque couvrant les deux cellules concernées. Nous comprenons en effet qu'en partant de l'état 13 la personne n'aura pas la même chance d'atteindre une position dans la cellule 7 qu'une position dans la cellule 8.

C. *Algorithme implanté*

Nous disposons de tous les éléments pour présenter l'algorithme de notre filtre Bayésien d'approche par grille.

$MovementUpdate(Bel(x_i))$

Foreach x_i do $Bel(x_i) = \sum_{x_k} P(x_i / d, x_k) \cdot Bel(x_i)$

$SignalUpdate(Bel(x_i), y)$

$\eta = 0$

Foreach x_i do

$Bel'(x_i) = P(y / x_i) \cdot Bel(x_i)$

$\eta = \eta + Bel'(x_i)$

Foreach x_i do

$$Bel(x_i) = \frac{Bel'(x_i)}{\eta}$$

L'algorithme comporte deux fonctions appliquées à chaque étape. La première fonction à un rôle de prédiction, c'est à dire qu'elle redistribue les probabilités de présence obtenues à l'étape précédente selon le modèle de déplacement de l'utilisateur choisi. De manière imagée, cette fonction produit une explosion de la probabilité dont les débris iront se répartir sur les cellules voisines (dans un rayon de 1.39 $m.s^{-1}$ ici). La seconde fonction vient immédiatement corriger cette prédiction en se basant sur l'empreinte signal enregistrée qu'elle compare à la carte des signaux. Ainsi, nous déterminerons toutes les secondes la fonction *Bel* pour l'ensemble des positions x_i. Pour une illustration de principe unidimensionnelle, il convient de se référer à l'annexe 2.

IV. Architecture de l'application

A. *Rappel des contraintes*

A terme nous envisagions une application embarquée sur une plateforme de type Gumstix dotée d'une carte d'extension Wi-Fi. En raison des délais qui nous ont été imposés et du temps nécessaire à la familiarisation avec de telles cartes, nous avons recadré l'objectif initial en décidant que la maquette de l'application tournerait sur un ordinateur PC portable équipé d'un carte Wi-Fi Netgear. Nous avons décidé d'utiliser principalement le langage Java afin de développer rapidement le module d'interface graphique cliente et le module

algorithmique. Notons toutefois que pour une utilisation sur système embarqué, il serait probablement indispensable de traduire le dernier module en C. L'objectif est évidemment d'optimiser autant que possible le temps d'exécution de l'algorithme car l'application finale devra être capable de fournir une estimation de position, au pire des cas toutes les secondes.
Habituellement la création d'un logiciel embarqué s'effectue sur un poste de travail dont l'architecture matérielle est souvent différente de celle du système embarqué cible. On parle alors de développement croisé. Il est alors nécessaire de créer une chaîne de compilation croisée qui permet de produire par exemple du code non-x86 sur un PC x86. Sous Linux, la chaîne de compilation GNU utiliserait les briques suivantes : le compilateur gcc, les outils annexes regroupés dans le paquetage binutils du type assembleur, éditeur de liens, et enfin la librairie C GNU-Libc. Le paramétrage de la chaîne de compilation est une étape très minutieuse et le novice préfèrera se rabattre sur des outils entièrement dédiés à cette tâche. Nous citons à titre d'information les deux plus importants qui sont tous les deux diffusés sous licence GPL : l'outil ELDK (Embedded Linux Development) existant sous forme de paquetage binaire et l'outil CROSSTOOL plus complexe de par l'inexistence de support d'installation (l'outil CROSSTOOL met à disposition de l'utilisateur un ensemble de scripts lui facilitant la mise en place d'une chaîne de compilation croisée personnalisée).
Pour terminer sur les contraintes, il ne faut pas oublier qu'une gestion optimale de l'énergie doit être réalisée pour garantir une autonomie suffisante et donc une utilisation agréable par l'utilisateur final. Nous n'avons qu'effleuré le problème lorsqu'il s'est agit de choisir entre une localisation par mesure de temps (solution active car la carte devait envoyer des requête ICMP au point d'accès en continu) ou par écoute du réseau (solution passive dans la mesure où la carte Wi-Fi ne fait qu'intercepter les paquets). Lors de nos tests, il s'est en effet avéré que la solution active entraînait une décharge extrêmement rapide de notre batterie d'ordinateur portable et qu'il était donc impensable de l'envisager sur un système embarqué. Selon nous, deux études plus poussée devraient être faites: une sur la consommation exacte en énergie de la carte Wi-Fi et une autre sur la complexité de l'algorithme qui détermine un ordre de grandeur sur le nombre d'opérations réalisées par le processeur et se retrouve donc liée à un problème de consommation. Nous axerons les parties qui suivent sur la conception et les évolutions des trois grands composants de notre application, à savoir : un sniffer C, un module algorithmique, un module d'interface graphique. Etant donné que chacun de ces modules a été développé indépendamment des autres, nous insisterons sur le travail d'intégration nécessaire à la fusion des parties entre elles. Ce sera l'occasion de mettre des mots sur la manière dont nous avons pensé l'interfaçage.

B. *Développement du sniffer*

Le sniffer constituait en quelque sorte la clef de voûte de notre application. Il devait d'une part nous alimenter en données pour la construction *offline* de notre carte du signal et d'autre part alimenter en temps réel le module algorithmique *online* qui estimerait régulièrement la position de l'individu. Bien qu'ayant parcouru Internet pour trouver la perle rare, nous n'avons pas réussi à trouver un sniffer adapté à nos besoins c'est à dire facilement réutilisable et adaptable. Pour commencer nos premières expériences sur l'analyse du signal, nous nous sommes donc tournés vers une solution de remplacement mais non viable à long terme. Il s'agissait d'effectuer des mesures au moyen de logiciels tels que Netstumbler sous Windows ou Ethereal sous Windows/Linux puis d'ensuite parser le fichier dumpé. Très vite nous avons compris les limites d'une telle approche. A titre d'exemple, un scan sous Ethereal de dix minutes conduisait à un fichier XML de plus de 70 Mo. Le parsing d'un tel fichier nous demandait environ 3 minutes d'attente pour récupérer les quelques champs de données qui nous intéressaient. La décision fût donc prise de développer un sniffer en C sous Linux. Le choix de Linux tient au simple fait que les sources des outils dédiés à la gestion du Wi-Fi sont librement consultables (driver Madwifi, Wireless Tools). En revanche la décision de programmer en C ne relève pas réellement d'un choix mais bien d'une nécessité dans la mesure où les librairies à utiliser étaient toutes codées en C (néanmoins nous tirons beaucoup d'enseignements et de plaisir de ce langage que nous avons trouvé hautement plus subtile mais nettement moins intuitif que Java).

Pour aborder le problème, nous avons commencé par lire le code source de certains des Wireless Tools qui nous semblaient intéressants pour la mesure du signal. Le « Tool » qui nous a servi de point de départ a été *iwlist*. Celui-ci permet de scanner les différents points d'accès à portée et d'en lister les caractéristiques. Sur la base de son code source notre premier sniffer vît le jour. Toutefois sa vie ne fût que très courte dans la mesure où nous nous sommes aperçu que pour fonctionner, notre sniffer devait nécessairement attendre que la carte Wi-Fi soit mise en mode *Managed* (mode actif). Or rappelons-nous que nous visions une solution totalement passive, c'est à dire en mode *Monitor*. Suite à ce premier échec, nous avons alors décidé de nous pencher sur les codes sources de deux célèbres sniffers, Ethereal et Kismet, tous deux fonctionnant en mode *Monitor*. L'étude des codes sources nous a guidé sur les traces d'une librairie essentielle : la librairie *Libpcap*. Elle permet la capture dynamique de paquets sur une interface réseau sélectionnée. Précisons aussi qu'elle est rattachée originairement au projet du célèbre utilitaire *tcpdump*. Doté de cette nouvelle librairie, la

grande difficulté fût de résoudre des problèmes d'éditions de liens lors de la compilation avec *gcc*. Il nous a fallu en effet identifier clairement d'une part les bons fichiers header de déclaration de fonction, d'autre part les librairies nécessaires empruntées aux Wireless Tools et à Libpcap, pour compiler notre sniffer C avec la commande *gcc* adaptée. L'écriture de la seconde version s'est faite entièrement from scratch sans reprise de code existant.
Une fois dotés de cet outil, nous avons construit nos premières cartes du signal et simulé des flux de mesures de signaux à partir de fichiers dumpés. Les flux de données étaient injectés dans le module algorithmique de façon à donner un aspect temps réel à l'application. De cette façon nous pouvions tester et analyser les calculs de probabilités au sein du module. Nous reparlerons plus bas dans le rapport de la troisième évolution du sniffer qui a permis son interfaçage avec le module algorithmique écrit en Java.
Pour le moment nous garderons à l'esprit que le sniffer nous permet d'enregistrer une trace de toutes les valeurs du signal sur une période de temps donnée et à une position donnée. Concrètement, si l'on se trouve en position $(0;1)$, le sniffer génèrera un fichier de dump nommé *X0Y1*. Nous avons conçu un petit programme Java qui nous permettait de synthétiser l'ensemble de ces fichiers bruts afin de ne récupérer que la moyenne et la variance des signaux pour chaque position. C'est pourquoi nous parlerons plus tard de fichier de synthèse.

C. Développement du module algorithmique

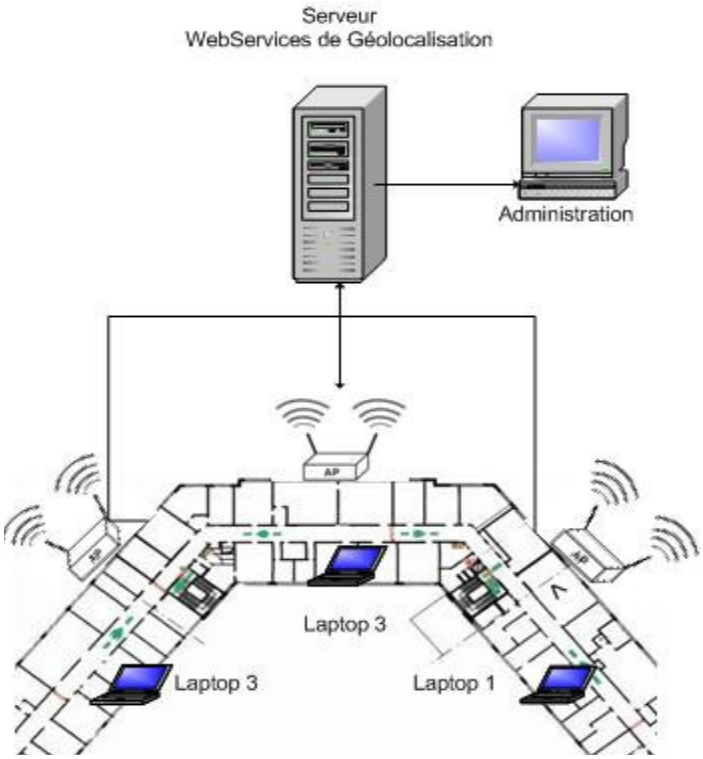
Dans cette partie nous ne revenons pas sur les aspects théoriques de l'algorithme déjà abordés mais présentons l'architecture en terme de diagramme de classes du module en question.
On distingue dans le module trois classes majeures. La première est la classe *Map* qui prend en charge la construction de la carte des signaux à partir du fichier de synthèse des signaux généré en phase *offline*. A la lecture de ce fichier des objets de type *AccessPoint* sont créés, chacun disposant d'un objet de type *Grid* qui n'est autre qu'un tableau stockant les mesures *offline* du point d'accès en question. Notons que pour les positions où nous ne disposions pas de mesure, il a fallu fixer une valeur par défaut qui dans notre cas était de -110 dBm (sachant que -100 dBm est la valeur seuil de sensibilité de la carte). La classe *Map* sera ensuite interrogée par l'objet *Belief*.
Les deux autres classes majeures sont *EstimationCalculator* et *Belief*. *Belief* nécessite à sa construction le passage d'une *Map* de base et crée une grille *Grid* pour le stockage des probabilités d'estimations. Les deux méthodes *movementUpdateBelief()* et

signalUpdateBelief() se chargent de mettre à jour la carte des probabilités. Ces mises à jour sont demandées par *EstimationCalculator* qui gère l'aspect dynamique de l'algorithme et la diffusion de l'information d'estimation de position. Notons que l'objet *Belief* fait appelle à la classe *CDN_Normal* lorsque des calculs de probabilités sur la loi normale sont demandés.

D. *Développement du module graphique*

Dans le cadre du développement et de l'implémentation d'algorithmes de géolocalisation, notre approche fut de séparer en deux programmes totalement indépendants les fonctions de calculs de la position et celles d'affichage de la position calculée.
De cette manière nous nous assurons une grande liberté dans l'évolution ainsi que dans l'architecture de notre application. En effet même si dans la version que nous avons jointe à ce dossier les deux modules sont sur la même machine, on pourrait parfaitement délocaliser le calcul de la position sur un serveur, qui proposerait un service de géolocalisation.
Cette solution aurait, entre autres avantages de décharger l'objet ubiquitaire de la lourde charge de calcul inhérente aux filtres Bayésiens. Aussi, cette architecture permettrait de réaliser un certain nombre de services comme :

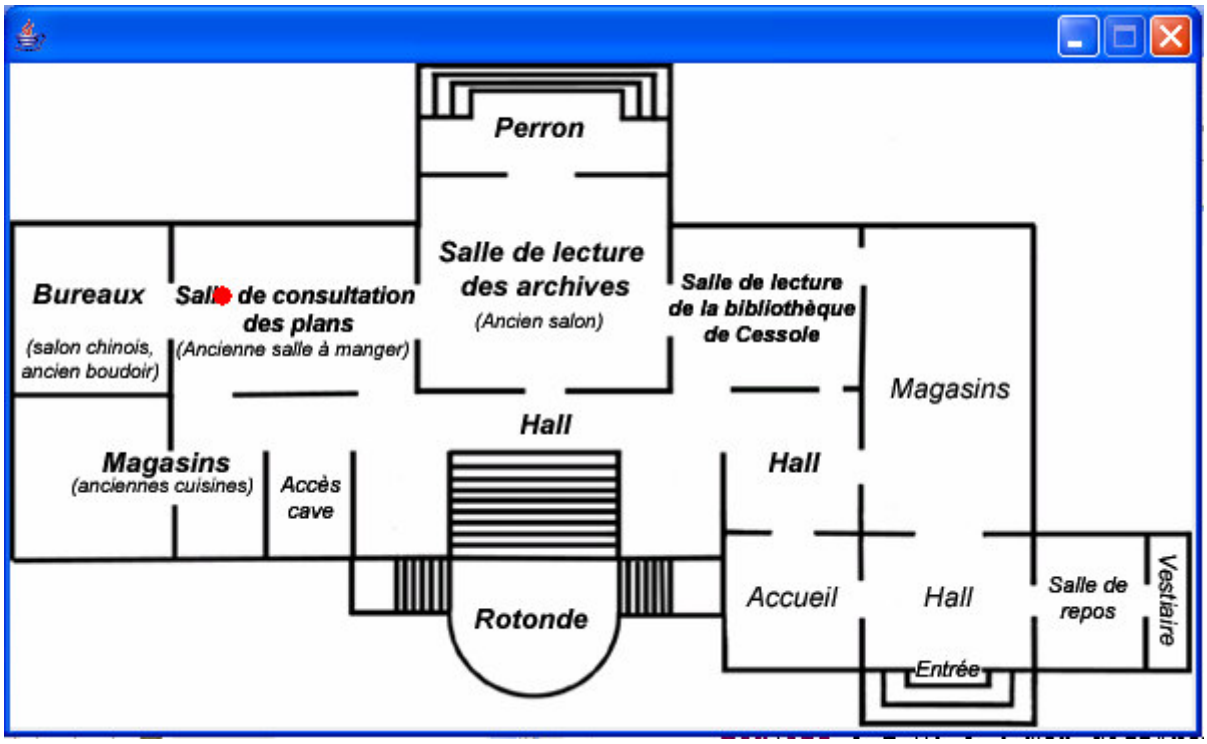
- La diffusion d'informations spécifiques en fonction de la zone dans laquelle se trouve l'objet
- La connaissance de la dernière position calculée par le serveur pour un client
- De permettre la création de tout un panel de services périphériques, plugin pour Messenger, plus court chemin d'un point à un autre…



Pour revenir à l'affichage de la position, il nous fallait une application permettant l'ajout de modules facilement, en d'autres termes que le modèle ne soit pas impacté dans l'affichage.

1. Choix du module d'affichage

Deux solutions étaient possibles, développer une application « maison » ou faire appel à un framework déjà développé. Dans notre cas, nous avons un peu fait les deux. En effet, dans un premier temps nous avons développé un package qui affichait une carte en arrière plan et qui déplaçait un point dessus (voir ci-dessous).



Ecran de la première version de notre éditeur de carte

Cependant nous nous sommes rapidement heurtés aux limites d'une architecture se prêtant peu à l'évolution et bloquant donc notre développement. Par exemple, la création d'un outil permettant le dessin de zones coloriables ou encore de mécanismes de gestion de calques aurait nécessité beaucoup trop de temps de développement.

Nous avons finalement préféré regarder si il n'existait pas des frameworks graphiques open source. Evidemment les productions sur ce sujet ne manquent pas, mais seules deux nous ont semblé bien documentées pour permettre une prise en main rapide. Ces deux frameworks sont :

- Le célèbre GEF (Graphical Editor Framework de Eclipse) http://www.eclipse.org/gef/
- Un homonyme de GEF mais produit entre autre par des étudiants de l'université de Californie et qui est notamment utilisé dans ArgoUML (un outil de dessin de digrammes UML) http://gef.tigris.org/

Même si au niveau du fonctionnement et de l'architecture les deux se ressemblent et utilisent les mêmes bases, après avoir suivi un tutorial pour les deux environnements, la deuxième option nous a semblé plus simple à manipuler, d'autant plus qu'elle disposait de beaucoup plus d'exemples commentés en ligne.

Revenons à la problématique du choix d'un framework graphique, plutôt que d'une solution maison. Les frameworks graphique du niveau de GEF sont le fruit du travail de beaucoup de

programmeurs, ils gèrent des événements complexes comme le drag'n drop ou encore les notions de calques et de transparence. Il sont robustes et souvent soutenus par des spécialistes de l'architecture logicielle.

Comprendre le fonctionnement de ce genre d'environnement et intégrer les différents patterns utilisés nous apporte beaucoup plus que la création de notre propre framework.

Aussi, pour mieux présenter le framework graphique utilisé dans notre application de géolocalisation nous allons illustrer le pattern MVC (Model Vue Controller) en le présentant dans un premier temps et en détaillant sont implémentation dans le cadre de notre éditeur graphique.

2. Utilisation du Design Pattern MVC

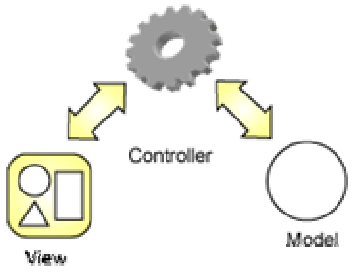
a) Présentation du pattern MVC

Tout d'abord un peu d'histoire :

Nous devons ce motif à Trygve Reenskaug, qui en 1979 travaillait sur Smalltalk dans les laboratoires de recherche Xerox PARC. A l'époque la notion même de design pattern n'existait pas.

Le motif de conception (Design Pattern) Modèle Vue Contrôleur (MVC) sert à optimiser le fonctionnement du logiciel en séparant le modèle de la donnée. En effet, le but est de séparer l'interface utilisateur de sa logique de contrôle.

Ce modèle comprend trois parties fondamentales (voir aussi le schéma ci-dessous) : le modèle, la vue et le contrôleur :



- Le Modèle : dans cette partie de l'application on met tout ce qui est relatif au comportement de l'application (traitements des données, interactions avec la base de données, etc). On y décrit aussi les différentes méthodes d'accès aux données ainsi que la structure des données.
- la Vue : contient les classes avec lesquelles l'utilisateur interagit en réalisant des événements. Le résultat des interactions de l'utilisateur est renvoyé par le modèle et est dénué de toute présentation ou mise en forme. On peut afficher plusieurs fois le même modèle à partir de plusieurs vues. La vue peut être conçue en html et interroger une application sur un serveur, c'est d'ailleurs

l'utilisation la plus courante du MVC. Cependant si cette architecture est courante sur Internet elle reste peu utilisée dans les logiciels.

- Le Contrôleur : il gère la relation entre la vue et le modèle, il récolte les événements effectués sur la vue et appelle les méthodes concernées. Il ne fait pas de traitement ni de modification sur les données.

En résumé, lorsqu'un client envoie une requête à l'application, celle-ci est analysée par le contrôleur, qui demande au modèle approprié d'effectuer les traitements, puis renvoie la vue adaptée au navigateur, si le modèle ne l'a pas déjà fait.

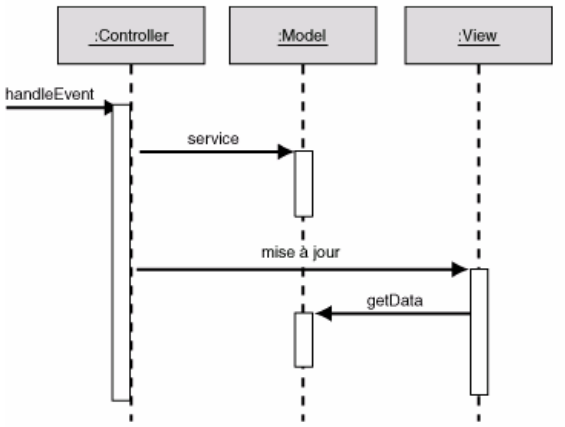


Diagramme de séquence après un événement

Un avantage apporté par ce modèle est la clarté de l'architecture qu'il impose. Cela simplifie la tâche du développeur qui tenterait d'effectuer une maintenance ou une amélioration sur le projet. En effet, la modification des traitements ne change en rien la vue. Par exemple on peut passer d'une base de données de type SQL à XML en changeant simplement les traitements d'interaction avec la base, et les vues ne s'en trouvent pas affectées.

b) Implémentation dans CardEditor

Dans les parties suivantes, nous allons présenter le fonctionnement de l'interface graphique de l'application. Nous retrouverons évidemment présentées les trois parties du modèle MVC au travers de trois parties éponymes.

Nous n'avons pas pour objectif de décrire ici de manière exhaustive le fonctionnement de l'éditeur graphique, mais plutôt de donner au lecteur les clés de déchiffrage nécessaires à une compréhension rapide du fonctionnement de l'éditeur.

Le package Utils n'est pas décrit dans cette partie car ce n'est qu'une collection d'outils aidant à réaliser des taches comme parser un fichier ou convertir un gif en jpg. Cependant, on aurait pu décrire des processus comme la sauvegarde mais malheureusement, ils ne sont pas totalement fixés dans l'état actuel de l'application.

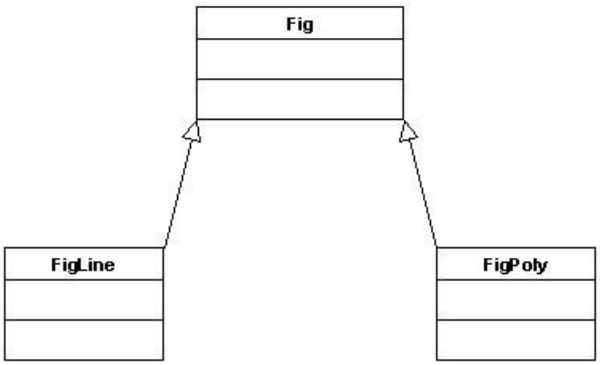
3. Historique de l'interface graphique

Le cahier des charges de l'application a évolué depuis ses débuts. En effet, suite à une divergence de point de vue, l'interface avait été développée afin de peindre des zones en fonction de la probabilité de s'y trouver.
A contrario, l'algorithme donne une position et la probabilité d'y être réellement, ce qui n'est que sensiblement différent. Le passage d'une vision de l'application à l'autre n'a donc pas été trop difficile.
Malheureusement, le développement de toute une partie de l'application a du coup été supprimé (Sauvegarde, gestion des zones…).

a) Le Modèle (Package Figures, Layer)

(1) Le package Figures

Les Figures sont le modèle des formes qui sont dessinées sur la carte à dessiner.
Même si la géolocalisation ne se matérialise que par un point qui se déplace sur la carte, le paramétrage de la carte ainsi que le développement de futures fonctionnalités nécessitent la réalisation d'objets qui définissent les zones. De plus, on pourrait facilement transformer les points en figures de couleurs et de formes différentes en fonction de paramètres comme la qualité de la mesure. De plus aujourd'hui la carte est considérée comme le fond d'une couche (Layer), mais une approche plus stricte du modèle voudrait que le fond, ainsi que les points matérialisant la position soient réintégrés dans la couche modèle.
Le digramme UML ci-dessous montre la structure du package Figures :



Le package figures comporte une superclasse abstraite Fig dont héritent toutes les objets à peindre comme les lignes `FigLine` qui servent à définir l'échelle de la carte et `FigPoly` qui sert à définir des zones.

b) Le Contrôleur (Package Mode, Commands, Selection, Layer)

La partie contrôleur du pattern MVC est implémentée au travers de quatre packages plus les classes `Editor` et `Globals` du package présentation. Dans cette partie nous allons expliquer le rôle de chacun de ces packages et comment ils s'articulent les uns autour des autres.

(1) La classe Globals

La classe `Presentation.Globals` est un singleton qui sert à stocker un certain nombre d'objets de manière à faciliter le dialogue entre les différents packages. En effet, pour récupérer une instance de editor par exemple il faut passer par Globals. La gestion des écouteurs attachés aux figures est aussi assurée par cette classe.
Dans le cadre de notre application Globals a particulièrement servi à la mise en place de drapeaux pour savoir si on est en mode *online* ou *offline*. Cette classe stocke aussi un bon nombre d'informations de configuration comme le numéro du port UDP, l'adresse de la carte en fond etc.

(2) Le package Layer

Le package Layer contient les différentes couches qui sont utilisées dans l'application. La classe abstraite `Layer` illustre parfaitement l'intérêt d'utiliser un framework graphique évolué. Chaque Layer hérite de la classe abstraite `Layer` où est défini tout un ensemble d'arguments qui permettent de gérer le processus de dessin de la couche.

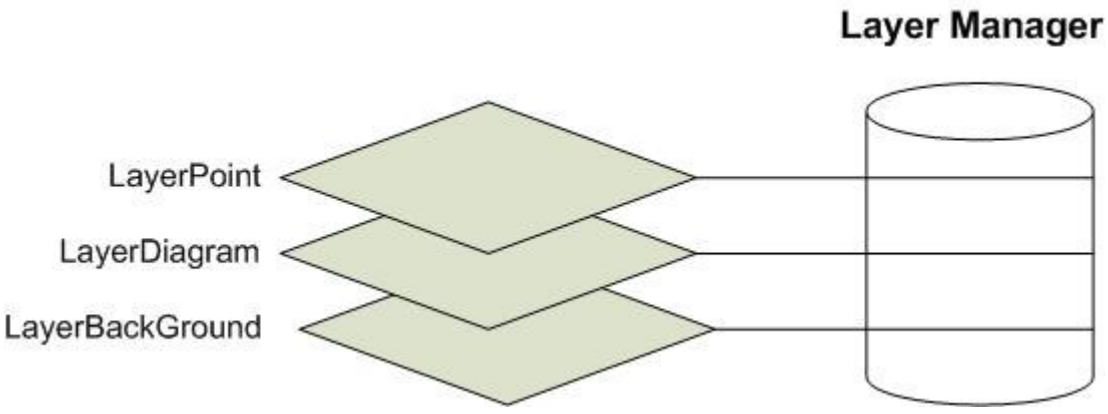
Un `Layer` est comme une feuille blanche qui peut contenir une partie de l'image à peindre ainsi que d'autres `Layers`. Grâce à cette classe on peut définir l'état d'un objet Layer comme grisé, caché…

Certaine sous-classe de `Layer`, comme la classe `LayerDiagram` servent à peindre des images, des figures ou encore contiennent d'autres `Layer`. Alors qu'à contrario des couches comme `LayerGrid` servent surtout à gérer certaines fonctionnalités comme le dessin d'une grille en arrière plan et surtout le magnétisme de l'arrière plan.

(3) Les classes Editor, LayerManager et SelectionManager

La classe `Editor` crée un éditeur dont le but est de manipuler des objets graphiques. L'éditeur est la pièce maîtresse de l'architecture de l'application. En effet, la classe Editor redistribue les événements aux autres classes comme le `LayerManager` ou le `SelectionManager` au travers de la classe `ModeManager`.

Chaque éditeur à son `LayerManager` qui gère les différents calques. Dans notre application, lors de la construction de l'objet `Editor` on crée un `LayerManager` structuré de la manière suivante :



Un `LayerManager` a une pile qui contient des calques ; dans notre cas seule la classe `LayerDiagramm` contient des objet de type `Figs` (qui représentent les zones sur la carte). Le Layer manager permet d'accéder facilement aux différentes Layer qu'il contient.
Quand des figures sont sélectionnées par un utilisateur, le `SelectionManager` crée un Objet `Selection` en fonction de l'action de l'utilisateur. Nous n'avons pas eu à modifier ce package dans le cadre de notre application. Nous ne détaillerons donc pas son fonctionnement.
Quand une figure change d'état suite à une action de l'utilisateur, la méthode `damageAll`() de l'objet `Layer` hôte de la figure est invoquée afin de repeindre l'intégralité du canevas. Ce signal est aussi utile pour rafraîchir l'éditeur après une modification de la position des points.

(4) Le package Commands

Le package commands contient toutes les commandes qui régissent les actions que l'on peut faire sur un éditeur. On peut citer l'auteur à ce sujet :
«*The editor serves as a command shell for executing actions in much the same way that a DOS or UNIX command shell executes programs. Each command can have a Hashtable of "command-line" arguments and also look at global variables (its environment)*"
On crée donc une instance d'un objet `Cmd` que l'on configure ou non à l'aide d'une Hashtable et pour lancer l'action on envoie un signal `doIt`(). On notera que la classe `Cmd` hérite de `AbstractAction` de la librairie Swing. On peut donc facilement ajouter des objets `Cmd` à des boutons. Ces derniers lanceront automatiquement un signal `doIt`() à l'objet Cmd qui leur est associé. On peut ainsi voir dans la classe `Presentation.Palette` :

```
public void defineButtons() {
     add(new CmdSetMode(ModeSelect.class, "regularMode"));
     ...
```

On remarque au passage que le package Mode est piloté par le package Command.
Dans ce package, nous avons rajouté trois classes :

- `CmdOffline` : Rend tous les objets sélectionnables, affiche les boutons de la Palette et met à jour le drapeau `isOnlineMode` de `Globals` à `false`.

- CmdOnline : Grise toutes les figures et cache la palette. Il lance aussi le thread de lecture après avoir mis le drapeau `isOnlineMode` à `true`. On lance ainsi la mise à jour de la position.

Nous avons aussi modifié un certain nombre d'objets déjà existants parmi lesquels :

- `CmdNew` : Gère la création d'un nouvel environnement de géolocalisation. Un nouveau répertoire est créé à l'adresse fixée par la chaîne de caractères `SavedProjectsDirectory` de la classe `Globals`. Ce répertoire contient les différentes informations nécessaires à notre environnement, nous y mettons la carte, des fichiers de dump et un fichier au Format VEC qui est créé lors de l'activation de `CmdSave`.
- `CmdSave` et `CmdOpen` : Gèrent l'enregistrement et la l'ouverture de sauvegardes spécifiques à notre application. Nous utilisons un format développer au Loria le format VEC pour sauvegarder les zones qui sont représentées par des polygones (classe `FigPoly`). Nous avons largement modifié les spécifications de ce format afin de permettre l'association de fichiers de dump avec chaque objet FigPoly sauvegardé.

(5) Le package Mode

Tout d'abord, nous allons expliquer le processus qui amène à passer dans un nouveau mode. Lors de l'activation de `CmdSetMode` :

- On crée une instance du nouveau mode en fonction des paramètres passés
- On initialise le mode en lançant un signal `init`()
- On met à jour la classe `Globals` en l'informant

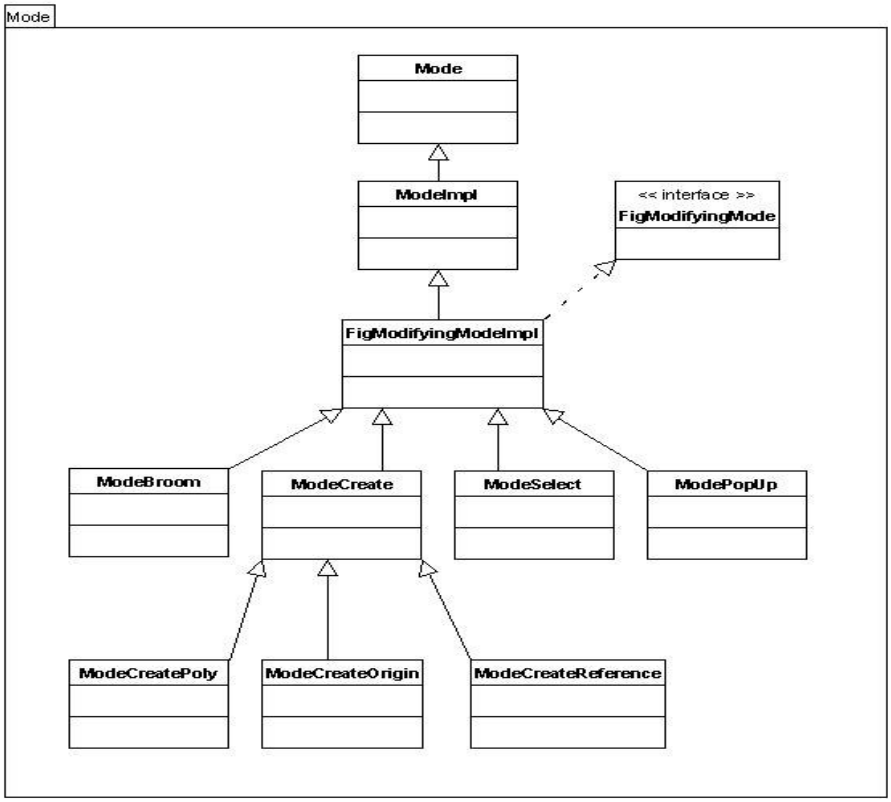
Tout nouveau mode créé est alors ajouté à la pile du `ModeManager` qui gère l'historique des modes. Tous les modes dans cette pile sont actif et plus le mode est en bas de la pile moins il a de chances de récupérer les évènements qui sont générés par l'utilisateur. Les modes se séparent en quatre types :

1. Les modes qui gèrent les modifications sur une figure (e.g. `ModeResize`)
2. Les modes qui gèrent la création d'une figure (e.g. `ModeCreate`)
3. Les modes qui gèrent la sélection (e.g. `ModeBroom`)
4. Les autres modes qui gèrent le dialogue avec l'utilisateur (e.g. `ModePopUp`)

Nous avons implémenté deux nouveaux modes :

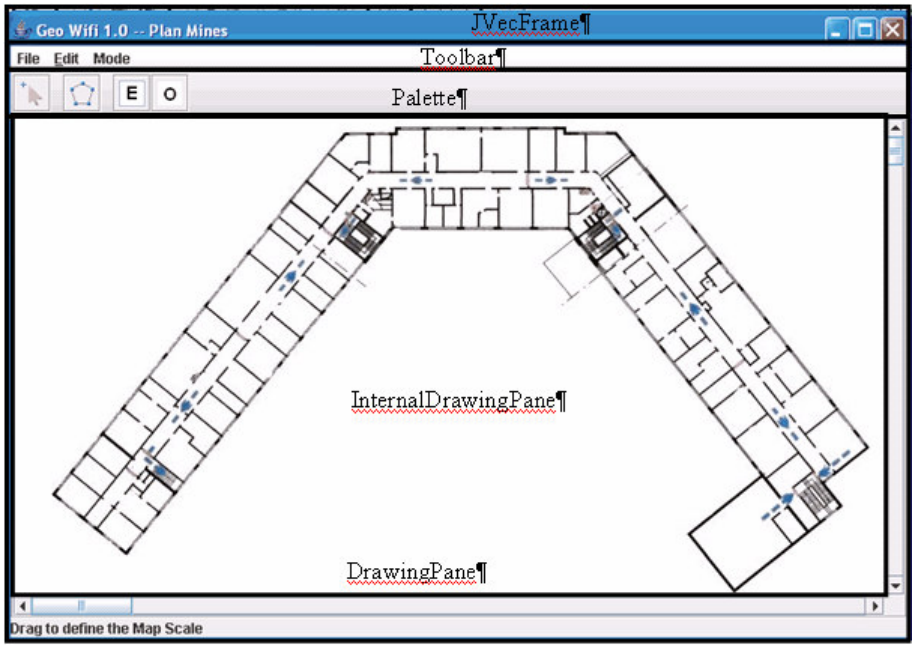
- `ModeCreateReference` : Paramètre l'échelle de la carte. Ce mode revient à créer une Ligne (`ModeCreateFigLine`), sans l'enregistrer dans la pile des figures de la couche `LayerDiagram`. On affecte à cette ligne une taille en mètre ce qui nous permet de définir l'échelle en pixel par mètre. Une fois cette échelle définie, on stocke la valeur dans `Globals` et on repeint le pupitre afin de mettre à jours la position des points.
- `ModeCreateOrigin` : Définit l'origine du repère de la carte qui nous a permis de réaliser nos mesures. Cela revient juste à récupérer la position de la souris lors d'un click et de la stocker dans `Globals`.

Le diagramme UML ci-dessous illustre la complexité du package mode :



On remarque la complexité de la gestion évènements sur les Figures. Cette partie du code n'a pas été modifiée, nous n'en avons pas eu besoin dans l'état actuel de l'interface.

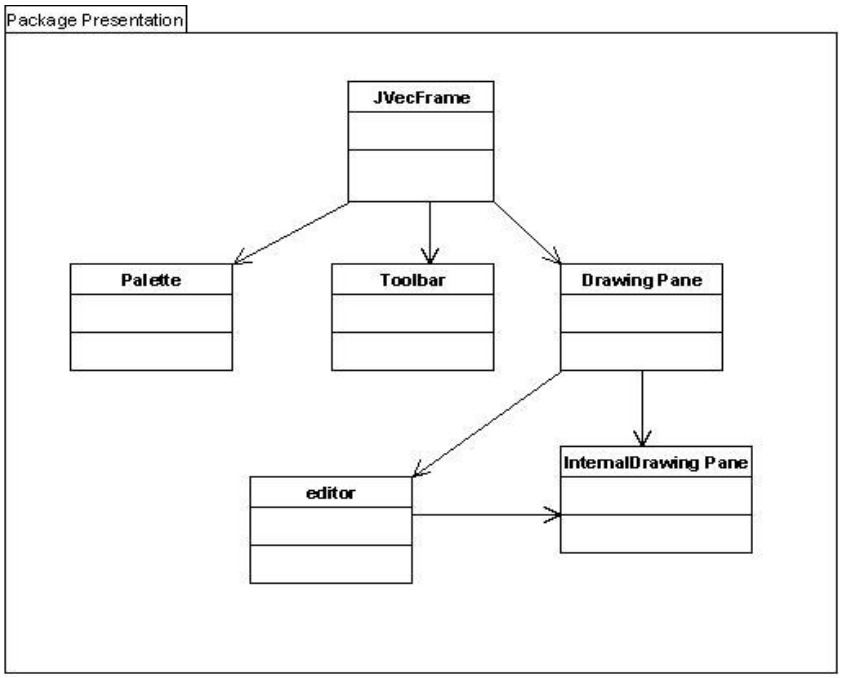
4. La Vue (Package Presentation)



Capture d'écran de l'application

Comme on peut le voir sur la capture d'écran ci-dessus, la structure même de l'affichage est assez sommaire. En effet, il n'y a pas de framework très évolué sur la vue, le principal objectif de cette partie du modèle est de mettre à disposition du contrôleur (classe `presentation.editor`) un `JComponent` duquel il récupèrera un objet `Graphics`. Cet objet servira de support pour dessiner la carte en arrière plan et le point qui symbolise l'emplacement de l'objet.

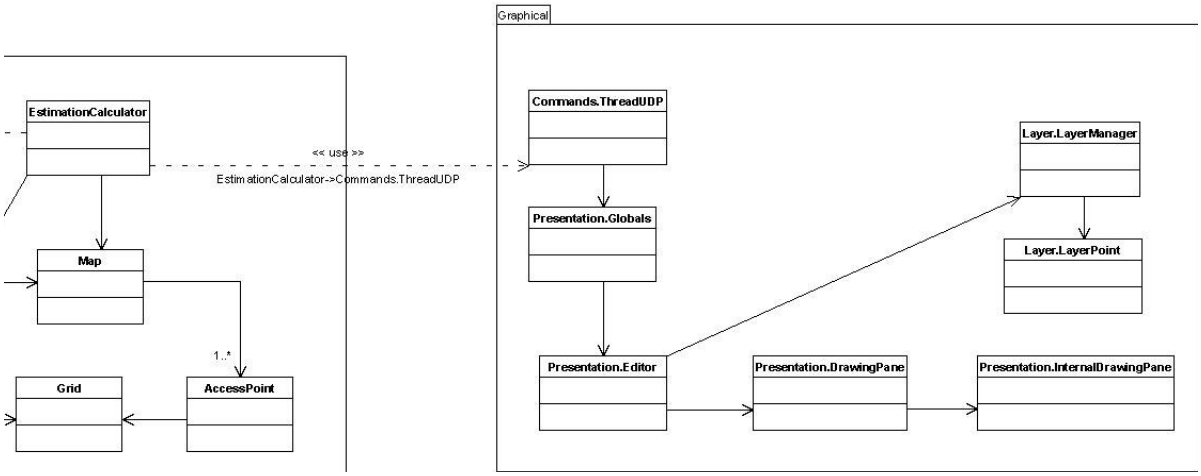
Le diagramme UML ci-dessous illustre de manière non exhaustive la structure de ce package :



`JVecFrame` possède une instance de `Toolbar`, de palette et de `DrawingPane` qui crée un objet editor dans son constructeur. Il passe ensuite un objet `InternalDrawingPane` qui hérite de `JPanel` par la méthode `editor.setJComponent(JComponent)` l'éditeur peut alors peindre l'intérieur du JPanel en fonction du model qui est à sa disposition.

5. Dialogue module Graphique/Calcul de la position

La version finale de l'interface donne accès à travers un *thread* à une méthode de mise à jour de la position du ou des points sur la carte. Le diagramme UML ci-dessous montre la structure de cette communication entre les deux applications :



L'interface d'affichage est dans l'état actuel de l'application totalement passive vis-à-vis de la partie calcul. Actuellement, l'utilisateur de l'interface graphique choisi à travers deux modes d'écouter ou non le port UDP et donc de mettre à jour l'affichage de la position.
L'application fonctionne donc suivant deux modes :

- *offline* qui sert à paramétrer la carte (échelle et origine du repère)
- *online* où l'application dialogue via un port UDP avec la partie statistique de l'application.

Cette architecture suppose bien sur que les deux modules sont sur la même machine, sinon l'interface affiche la position d'une autre machine qui héberge le *sniffer* et le module de calcul de la position.
Description du fonctionnement du mode *online* :
L'application est en mode *online* (drapeau dans la classe `Globals`), le Thread `ThreadUDP` écoute le port UDP et lit les informations correspondantes à une liste de points respectivement associés à des probabilités. Une fois la liste récupérée, le Thread met à jour la position des points sur la carte en récupérant via la classe `Globals` l'instance de la couche `LayerPoint` où sont dessinés les points.

E. *Interfaçage des modules*

Etant donné que les composants de l'application ont été construits de façon indépendante, il a été nécessaire de définir comment les modules allaient dialoguer entre eux pour se transmettre des informations.
Dans un premier temps, nous devions faire en sorte que le sniffer écrit en C puisse alimenter en continu le module algorithmique Java. Le problème fût donc de coder le « liant » qui permettrait de surmonter le problème d'incompatibilité des langages. L'idée fût de passer par des sockets TCP. Le choix du protocole de transport n'est pas anodin car TCP s'appuie sur une connexion en mode connecté ce qui nous garantira que tous les paquets transmis par le sniffer seront bien réceptionnés par le module algorithmique. Côté Java l'implémentation des sockets se fît sans grande difficulté au niveau de la classe *SniffAdapteur*. Côté C en revanche il nous a fallu consacré plus de temps pour saisir toutes les subtilités des sockets. Au final la communication entre ces deux composants fût établie et nous saisissons maintenant pourquoi les réseaux basés sur le protocole TCP/IP sont un moyen puissant d'intégration. Notons tout

de même que dans une application temps réel, le choix des sockets n'est peut être pas judicieux dans la mesure où celles ci sont amenés à faire appel à un niveau de ressources plus élevé qu'il ne l'est avec les tubes (*Pipes* en anglais) entre processus léger. Pour ne pas faire un usage démesuré des sockets au sein de l'application, nous avons décidé que le relais d'informations entre les threads *SniffAdapteur* et *EstimationCalculator* se ferait via des tubes.
Dans un second temps, nous nous sommes intéressés à l'intégration de la partie graphique cliente. Cette fois ci la décision fût prise de communiquer toujours par sockets mais cette fois ci en mode UDP (mode non connecté) pour éviter tout ralentissement au niveau du module algorithmique et simplifier la mise en oeuvre. A chaque nouvelle estimation de position, *EstimationCalculator* crée un datagramme contenant les cinq meilleures positions avec leur probabilité associée et le broadcast sur le réseau sans se soucier de sa bonne réception au niveau du module graphique. De son côté le module graphique écoute sur l'adresse broadcast et extrait les positions et probabilités des paquets UDP qu'il reçoit.

V. Résultats et perspectives

Les nombreuses expériences que nous avons réalisées pour mettre au point l'algorithme ont permis d'obtenir une précision de l'ordre de 2 mètres. Même si nous n'avons pas mis au point de véritable technique pour mesurer la précision en temps réel, nous avons présenté ce travail à notre tuteur qui peut certifier la qualité des estimations.
La majorité du développement qu'il reste à faire est en rapport avec la liaison des deux modules. En effet, on peut isoler voici quelques pistes de développement pour le futur :

- Paramétrage du module statistique via le mode offline de l'interface graphique Une application cliente qui se contenterai d'afficher la position et de charger des cartes
- Une application administratrice qui permettrait la configuration d'un éventuel serveur qui distribuerait entre autre chose un service de geolocalisation.
- Définir des zones correspondantes à certains messages publicitaires par exemple qui seraient envoyé au client qui passe dans la zone.

Notre travail s'apparente plus à une *proof of concept* et ouvre de belles perspectives en terme d'optimisation de l'algorithme, plus particulièrement au niveau du modèle choisi pour le déplacement de l'utilisateur. On pourrait aussi clairement mettre en place une chaîne de traitement qui raffinerait au fur et à mesure l'estimation brute émise. Parallèlement, l'application graphique pourra elle aussi bénéficier d'un raffinage fonctionnel et esthétique.

Conclusion

Ce projet que nous avions proposé personnellement nous a complètement immergé dans le monde du Wi-Fi. En effet nous avons découvert les multiples façons d'exploiter cette technologie sous Linux pour l'appliquer au thème de la géolocalisation. Même si aujourd'hui nous restons loin de l'objectif initial d'un journal de bord, nous disposons aujourd'hui d'une base solide de connaissances pour implémenter des services sur un objet ubiquitaire sur une plateforme Gumstix.

Les dernières expériences menées mi-février 2006 ont attesté de la qualité de l'algorithme et de l'interface graphique pour permettre le suivi d'un individu. Actuellement, de nombreuses entreprises comme Google ont déjà recensé plusieurs millions de points d'accès à des fins de géolocalisation. Pour l'instant la précision proposée est d'environ 20m, ce qui est n'est pas encore suffisant pour de nombreux services. Il existe un réel business à faire en s'appuyant sur des solutions telles que celle que nous avons explorée.

Annexes

1. Graphiques associés aux expériences

Expérience 1 :

- Variation du signal à 1m d'un point d'accès Belkin, sur une durée de 10min
- Evolution des proportions sur des intervalles de temps de 10sec
- Evolution des proportions sur des intervalles de temps de 2min

Expérience 2 :

- Netgear MIMO type point d'accès
- Netgear MIMO type routeur + point d'accès
- D-Link
- Belkin

Expérience 3 :

- Puissance du signal en fonction de la distance, couloir des mines, routeur D-Link
- Puissance du signal en fonction de la distance, extérieur, routeur Belkin (1)
- Puissance du signal en fonction de la distance, extérieur, routeur Belkin (2)

Expérience 4 :

- Puissance du signal en fonction de la distance, courte échelle, routeur Belkin

Expérience 5 :

- Puissance du signal en fonction de l'orientation, routeur Netgear

Puissance (en dBm)	Nombre de paquets	Proportion
0-10 sec		
-41	0	0,00%
-42	0	0,00%
-43	2	2,04%
-44	8	8,16%
-45	22	22,45%
-46	26	26,53%
-47	9	9,18%
-48	8	8,16%
-49	8	8,16%
-50	11	11,22%
-51	1	1,02%
-52	0	0,00%
-53	2	2,04%
-54	0	0,00%
-55	1	1,02%
	98	100,00%

Puissance (en dBm)	Nombre de paquets	Proportion
10-20 sec		
-41	0	0,00%
-42	0	0,00%
-43	3	3,03%
-44	14	14,14%
-45	27	27,27%
-46	26	26,26%
-47	3	3,03%
-48	8	8,08%
-49	12	12,12%
-50	6	6,06%
-51	0	0,00%
-52	0	0,00%
-53	0	0,00%
-54	0	0,00%
-55	0	0,00%
	99	100,00%

Puissance (en dBm)	Nombre de paquets	Proportion
20-30 sec		
-41	0	0,00%
-42	2	2,11%
-43	2	2,11%
-44	10	10,53%
-45	26	27,37%
-46	33	34,74%
-47	1	1,05%
-48	7	7,37%
-49	10	10,53%
-50	4	4,21%
-51	0	0,00%
-52	0	0,00%
-53	0	0,00%
-54	0	0,00%
-55	0	0,00%
	95	100,00%

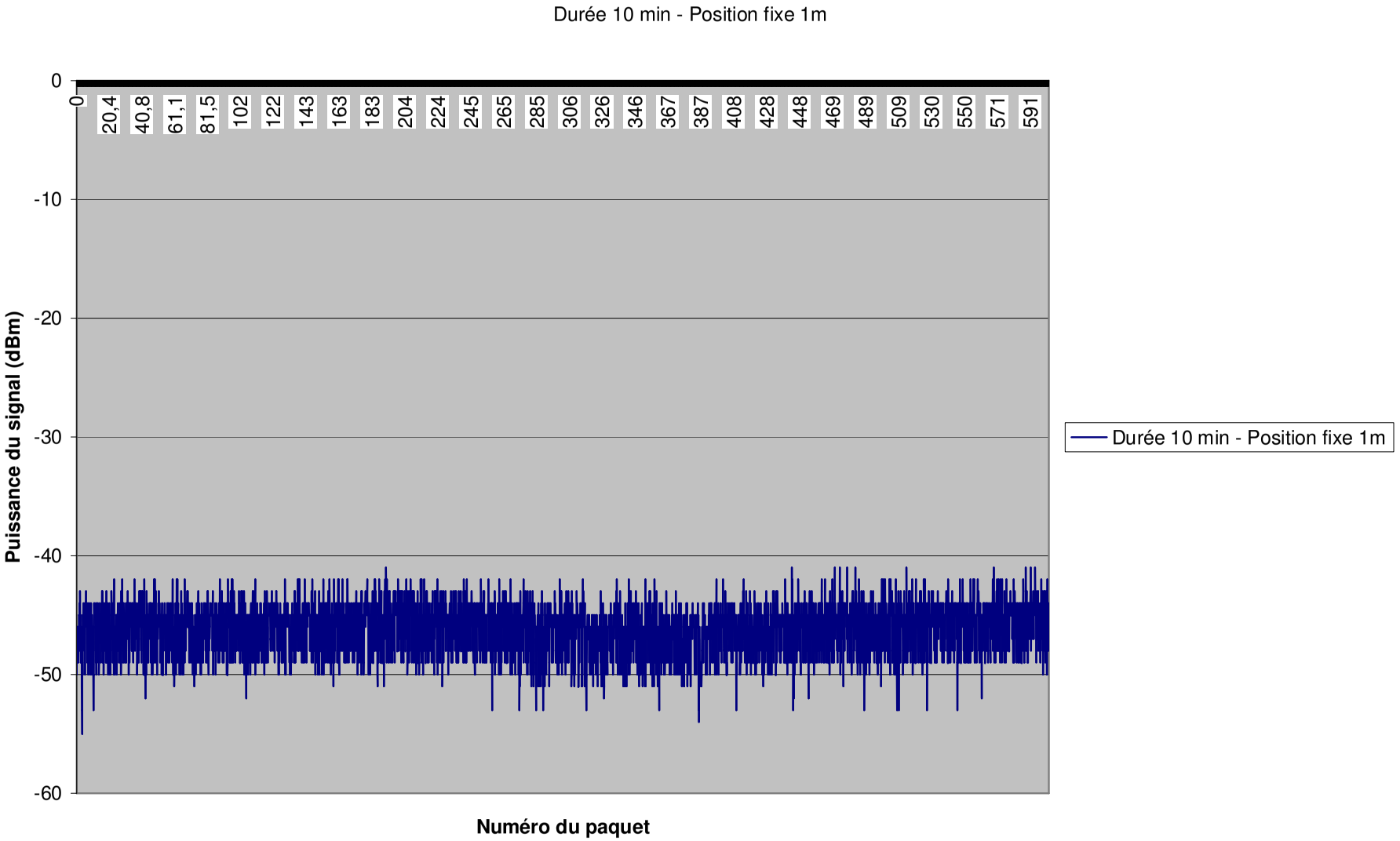
Puissance (en dBm)	Nombre de paquets	Proportion
30-40 sec		
-41	0	0,00%
-42	1	1,04%
-43	4	4,17%
-44	21	21,88%
-45	29	30,21%
-46	23	23,96%
-47	3	3,13%
-48	7	7,29%
-49	6	6,25%
-50	2	2,08%
-51	0	0,00%
-52	0	0,00%
-53	0	0,00%
-54	0	0,00%
-55	0	0,00%
	96	100,00%

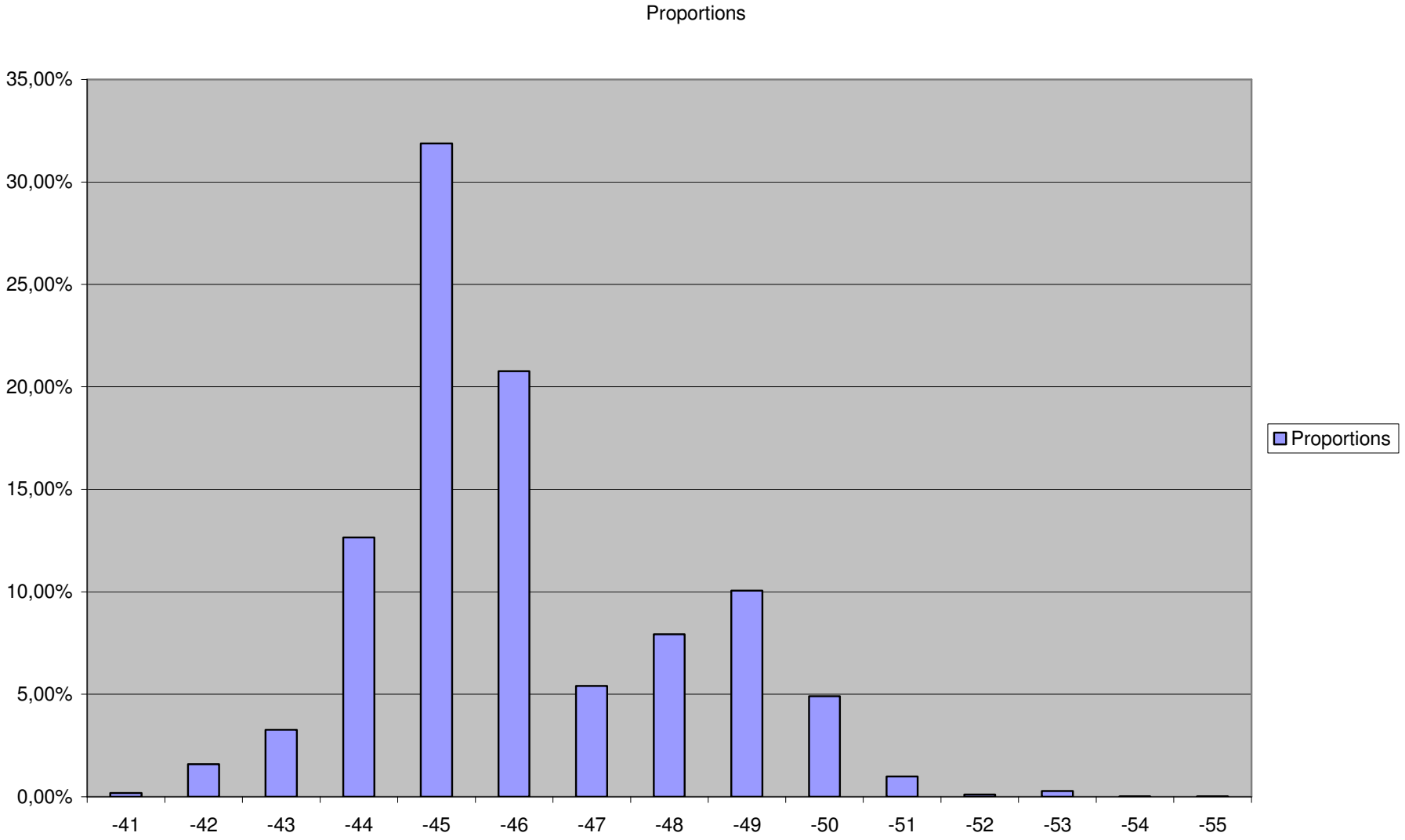
Puissance (en dBm)	Nombre de paquets	Proportion
0-2 min		
-41	0	0,00%
-42	15	1,28%
-43	36	3,07%
-44	130	11,10%
-45	385	32,88%
-46	278	23,74%
-47	39	3,33%
-48	88	7,51%
-49	122	10,42%
-50	70	5,98%
-51	3	0,26%
-52	2	0,17%
-53	2	0,17%
-54	0	0,00%
-55	1	0,09%
	1171	100,00%

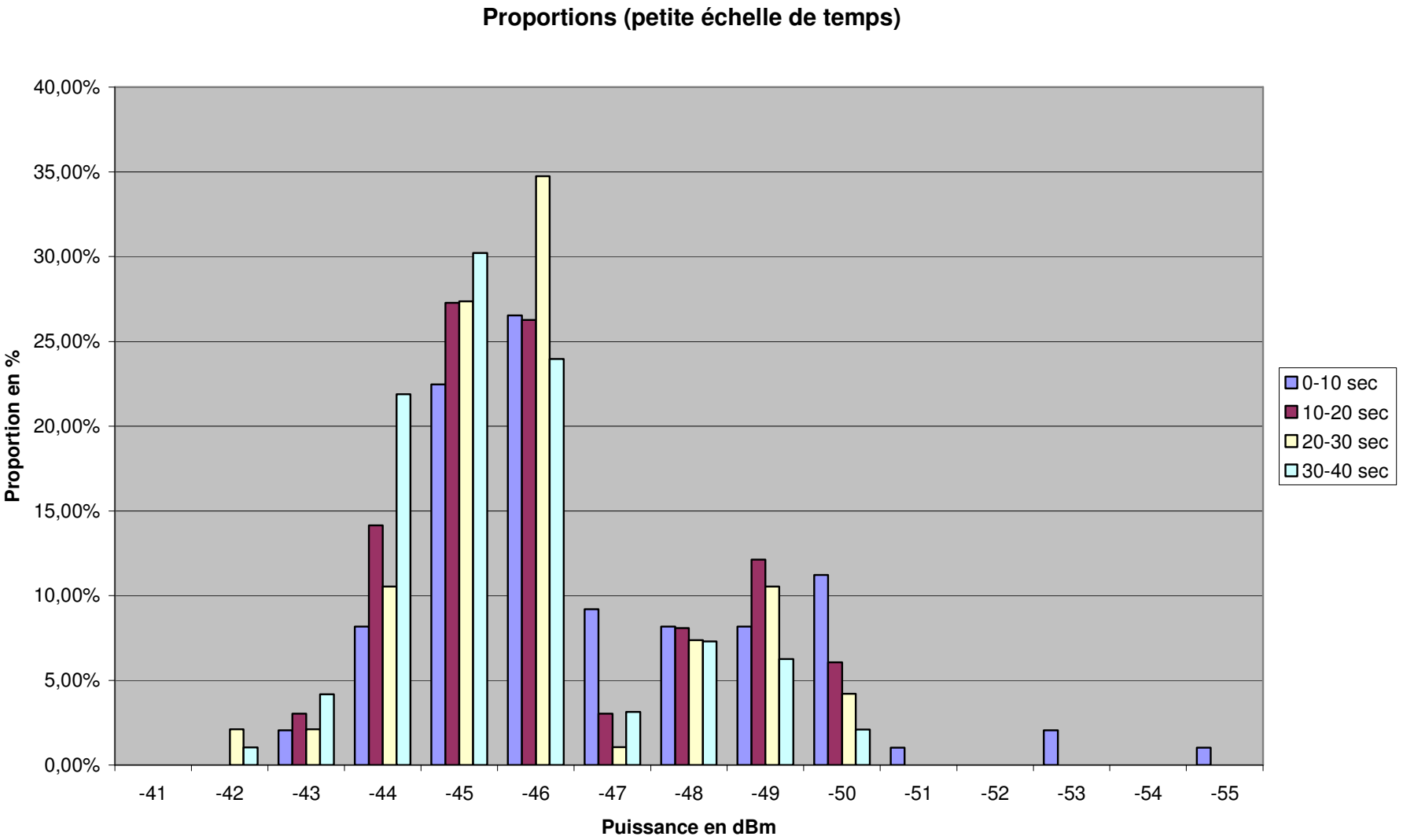
Puissance (en dBm)	Nombre de paquets	Proportion
2-4 min		
-41	1	0,09%
-42	28	2,39%
-43	62	5,29%
-44	170	14,49%
-45	486	41,43%
-46	129	11,00%
-47	33	2,81%
-48	109	9,29%
-49	120	10,23%
-50	31	2,64%
-51	4	0,34%
-52	0	0,00%
-53	0	0,00%
-54	0	0,00%
-55	0	0,00%
	1173	100,00%

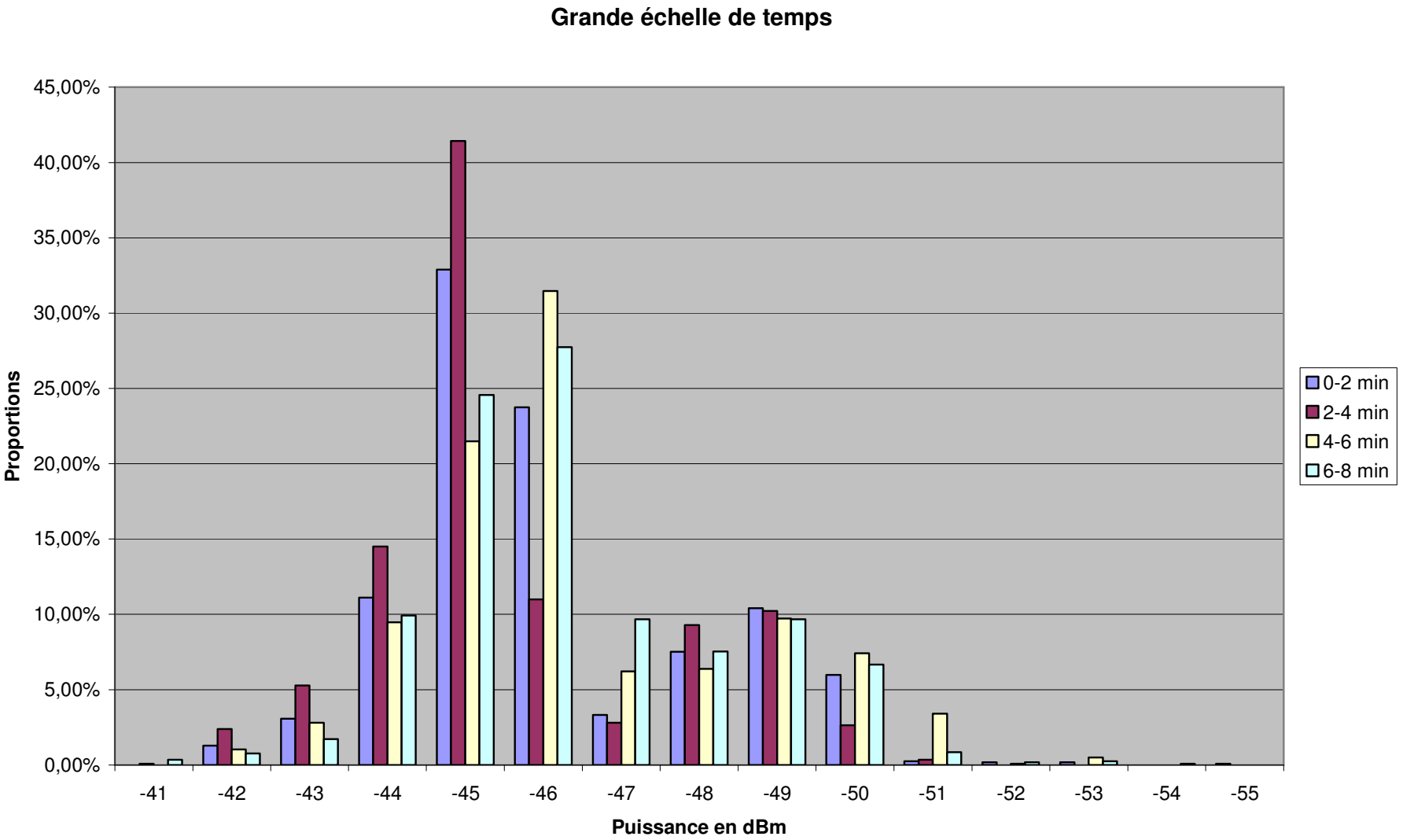
Puissance (en dBm)	Nombre de paquets	Proportion
4-6 min		
-41	0	0,00%
-42	12	1,02%
-43	33	2,81%
-44	111	9,46%
-45	252	21,48%
-46	369	31,46%
-47	73	6,22%
-48	75	6,39%
-49	114	9,72%
-50	87	7,42%
-51	40	3,41%
-52	1	0,09%
-53	6	0,51%
-54	0	0,00%
-55	0	0,00%
	1173	100,00%

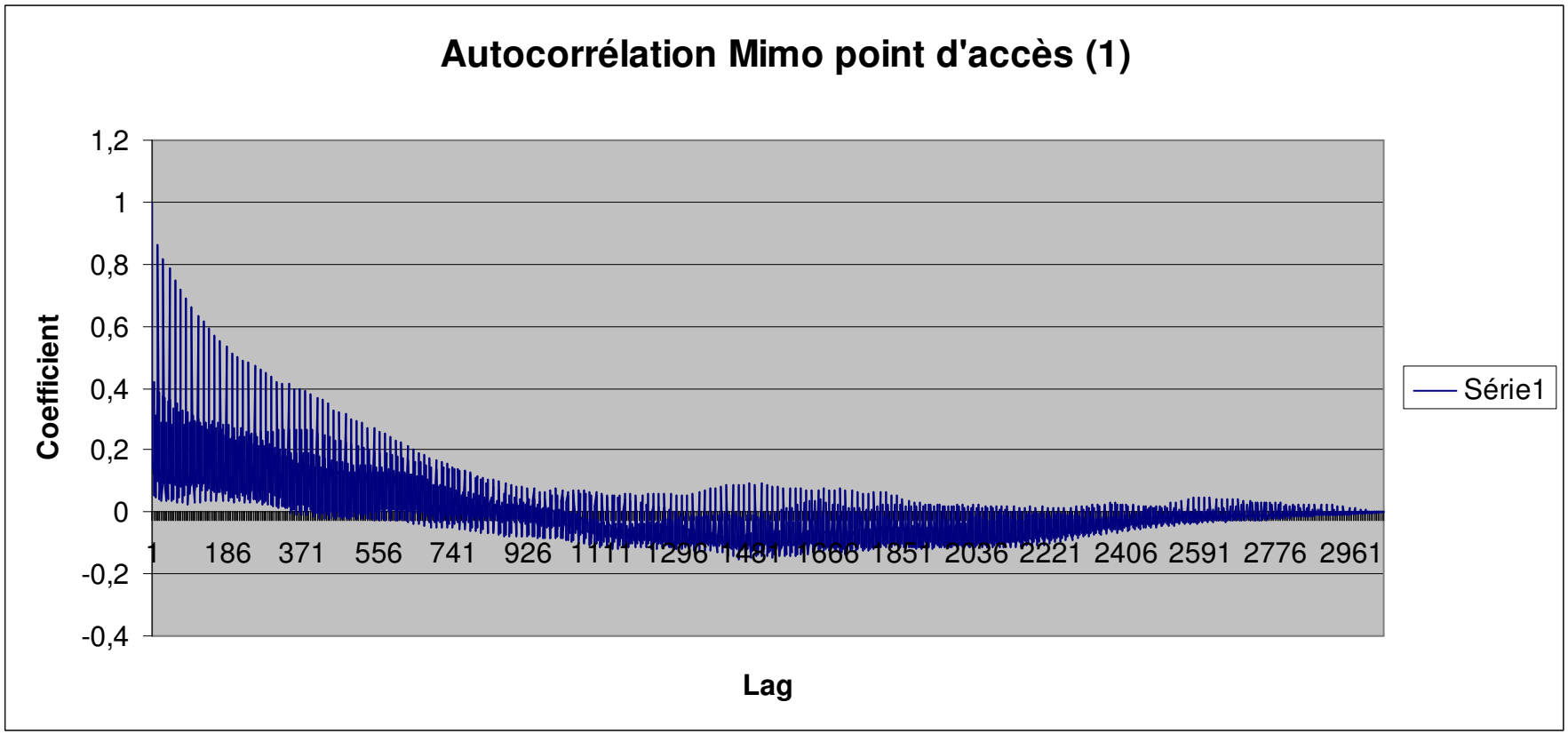
Puissance (en dBm)	Nombre de paquets	Proportion
6-8 min		
-41	4	0,34%
-42	9	0,77%
-43	20	1,71%
-44	116	9,93%
-45	287	24,57%
-46	324	27,74%
-47	113	9,67%
-48	88	7,53%
-49	113	9,67%
-50	78	6,68%
-51	10	0,86%
-52	2	0,17%
-53	3	0,26%
-54	1	0,09%
-55	0	0,00%
	1168	100,00%

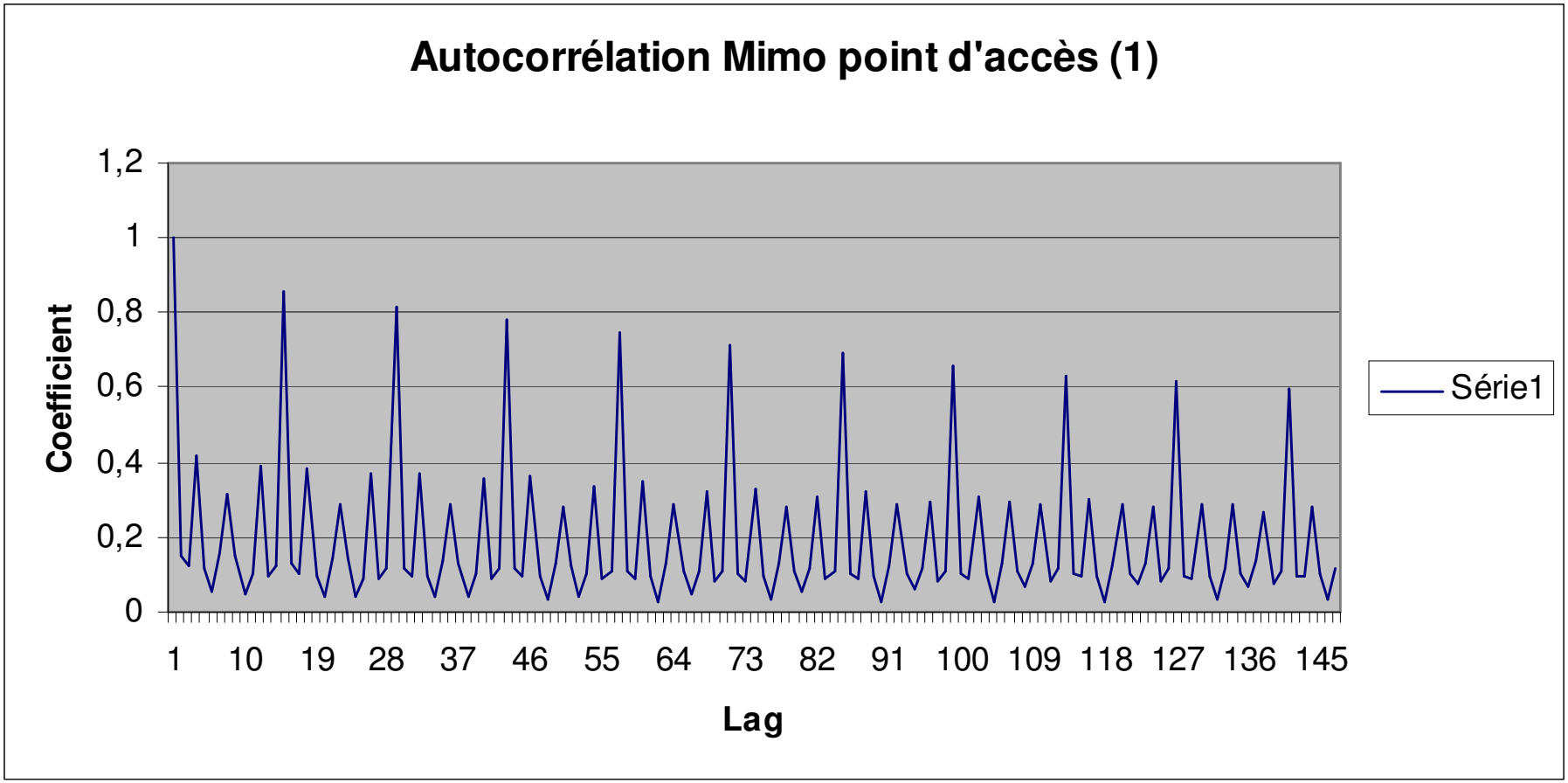

Durée 10 min - Position fixe 1m
0
-10
-20
-30
-40
-50
-60
Puissance du signal (dBm)
0
20,4
40,8
61,1
81,5
102
122
143
163
183
204
224
245
265
285
306
326
346
367
387
408
428
448
469
489
509
530
550
571
591
Numéro du paquet
Durée 10 min - Position fixe 1m

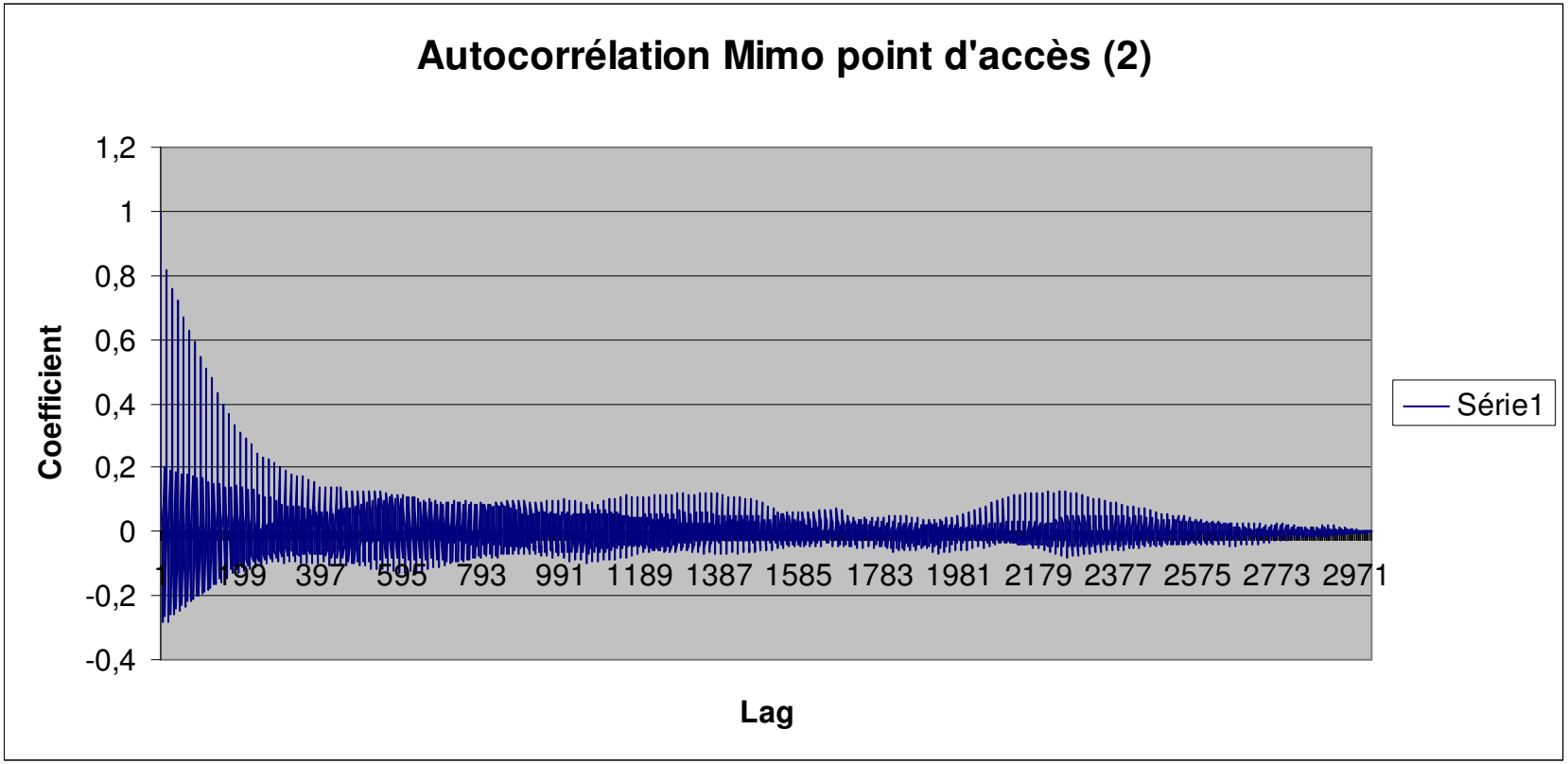

Proportions
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
-41
-42
-43
-44
-45
-46
-47
-48
-49
-50
-51
-52
-53
-54
-55
Proportions

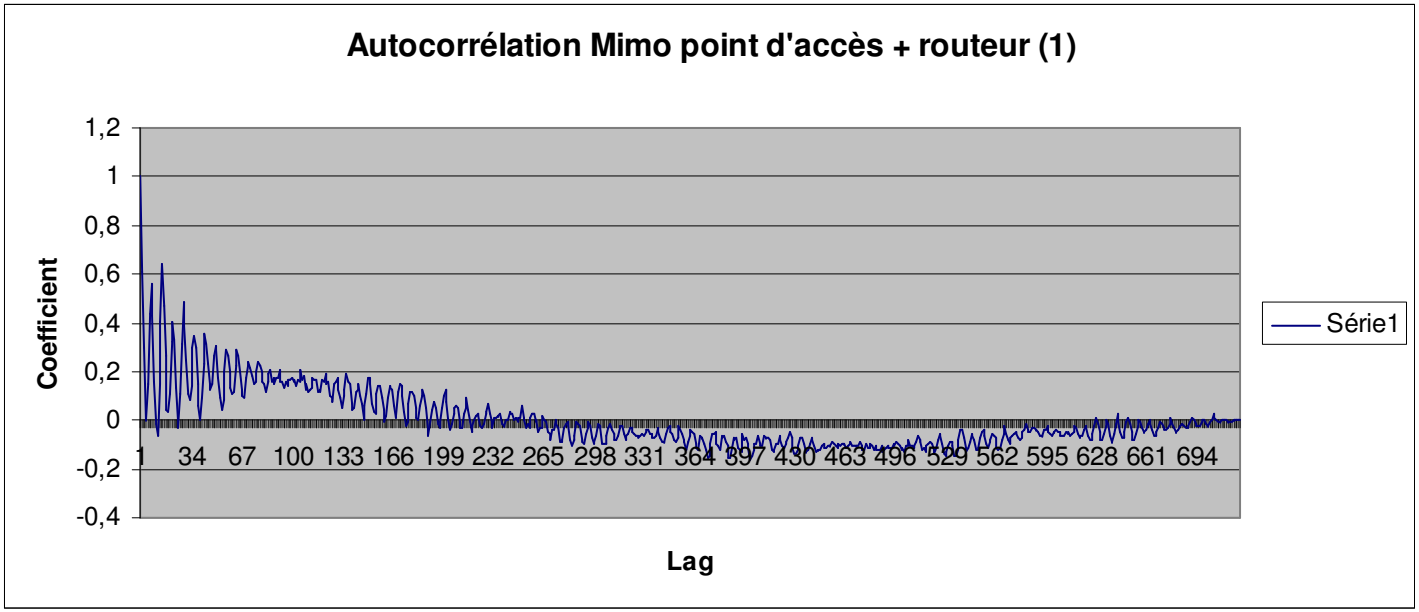

Proportions (petite échelle de temps)
40,00%
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Proportion en %
-41
-42
-43
-44
-45
-46
-47
-48
-49
-50
-51
-52
-53
-54
-55
Puissance en dBm
0-10 sec
10-20 sec
20-30 sec
30-40 sec

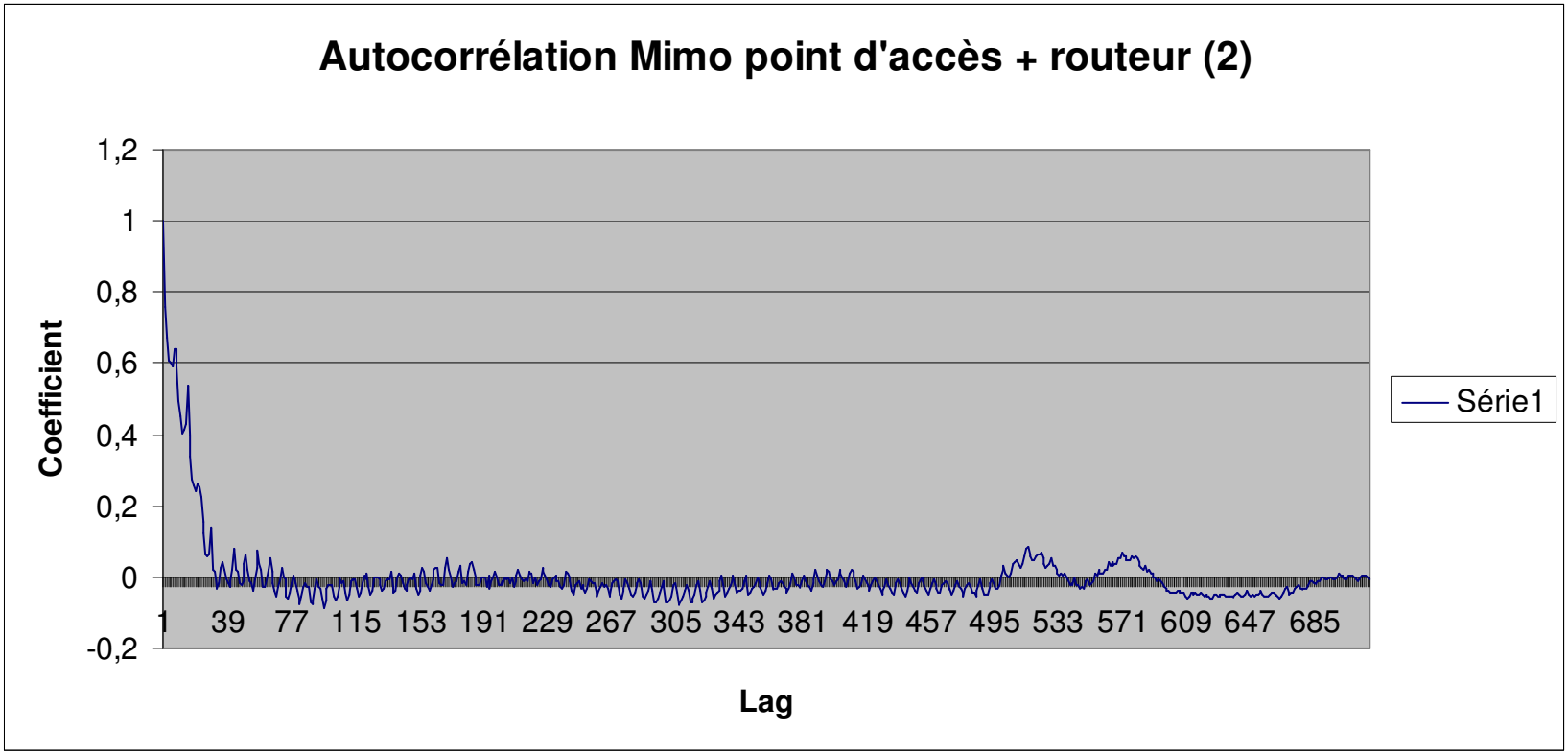

Grande échelle de temps
45,00%
40,00%
35,00%
30,00%
25,00%
20,00%
15,00%
10,00%
5,00%
0,00%
Proportions
-41
-42
-43
-44
-45
-46
-47
-48
-49
-50
-51
-52
-53
-54
-55
Puissance en dBm
0-2 min
2-4 min
4-6 min
6-8 min

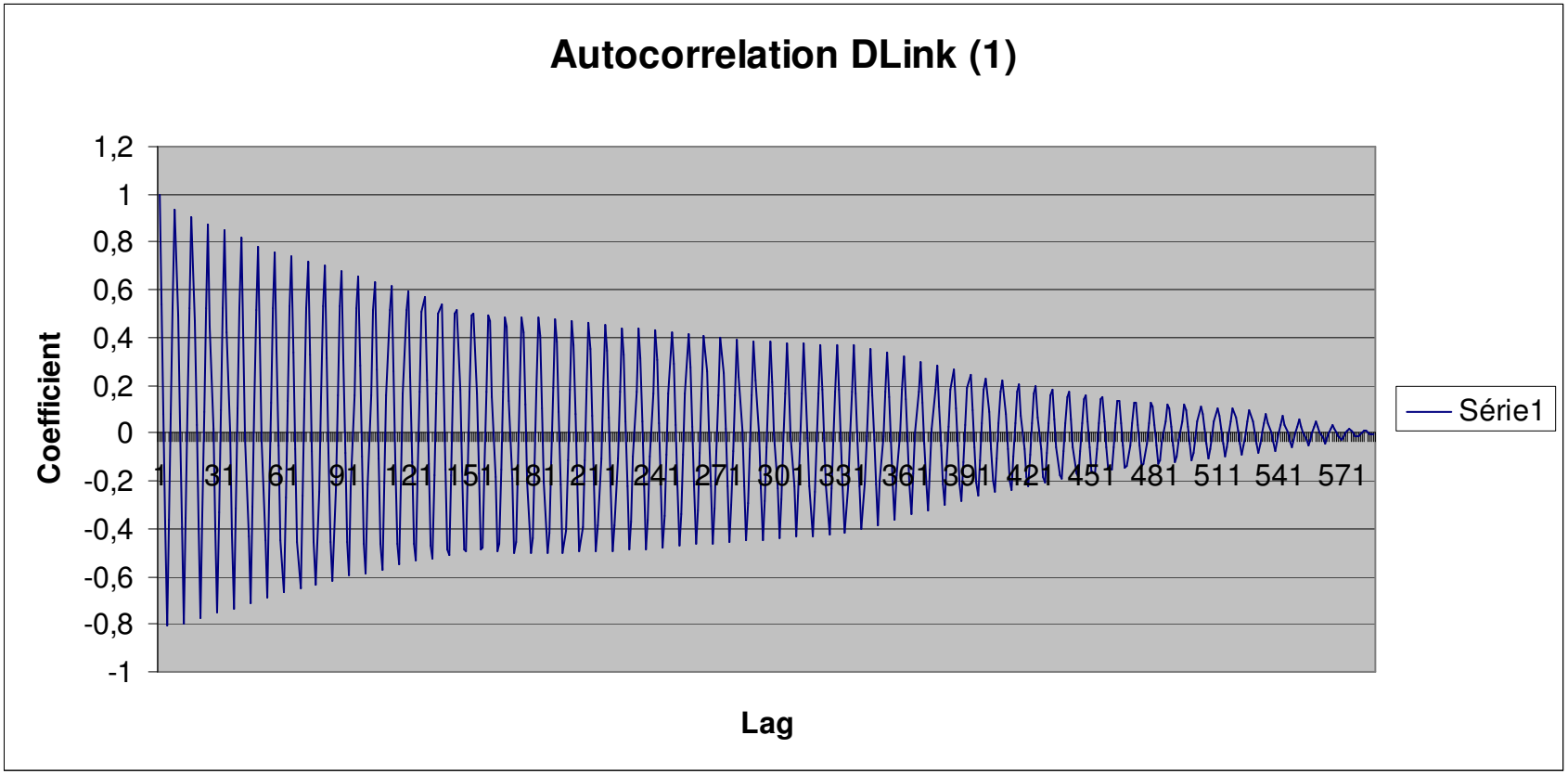

Autocorrélation Mimo point d'accès (1)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
1
186
371
556
741
926
1111
1296
1481
1666
1851
2036
2221
2406
2591
2776
2961
Lag
Série1

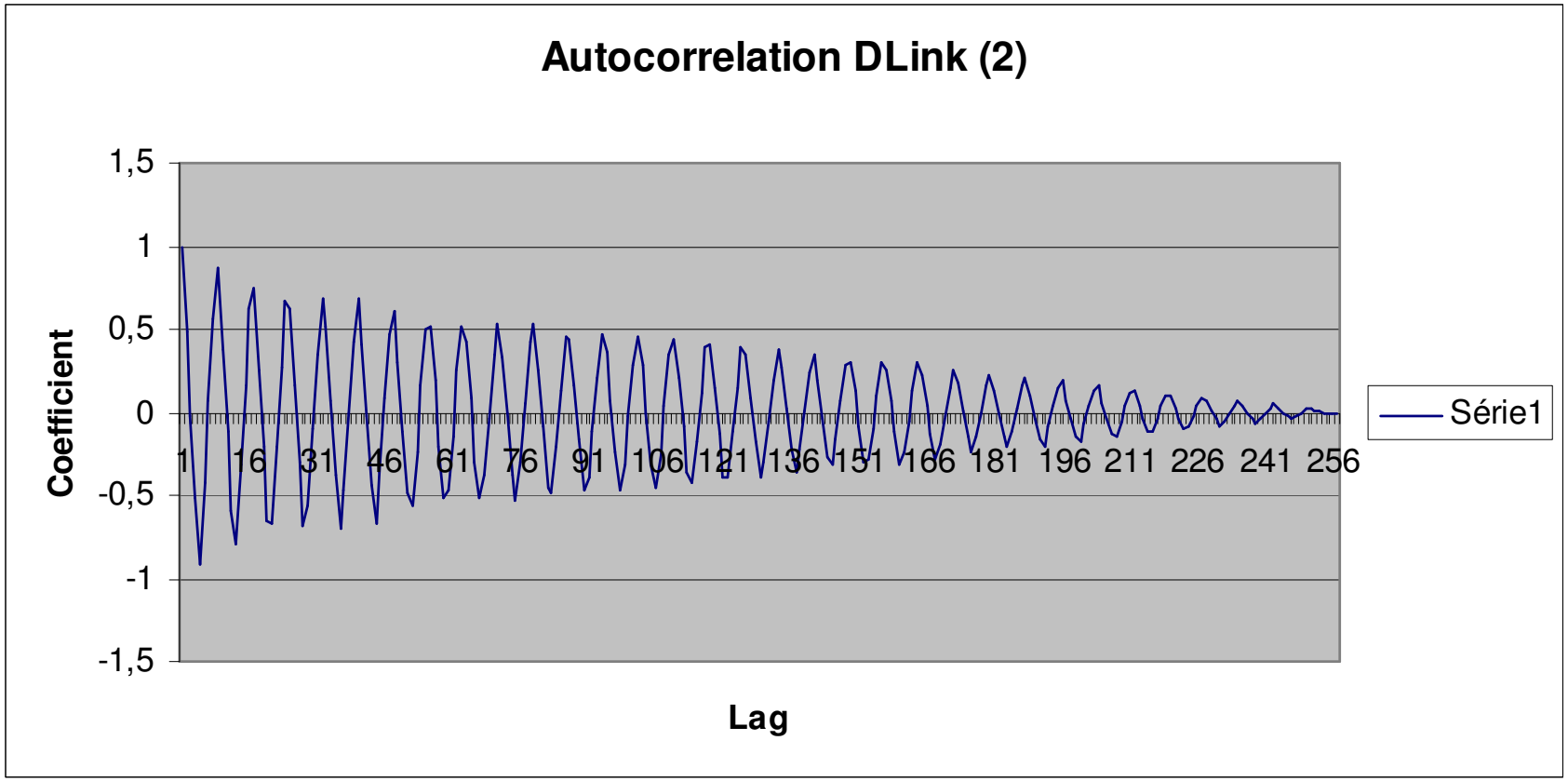

Autocorrélation Mimo point d'accès (1)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
1
10
19
28
37
46
55
64
73
82
91
100
109
118
127
136
145
Lag
Série1

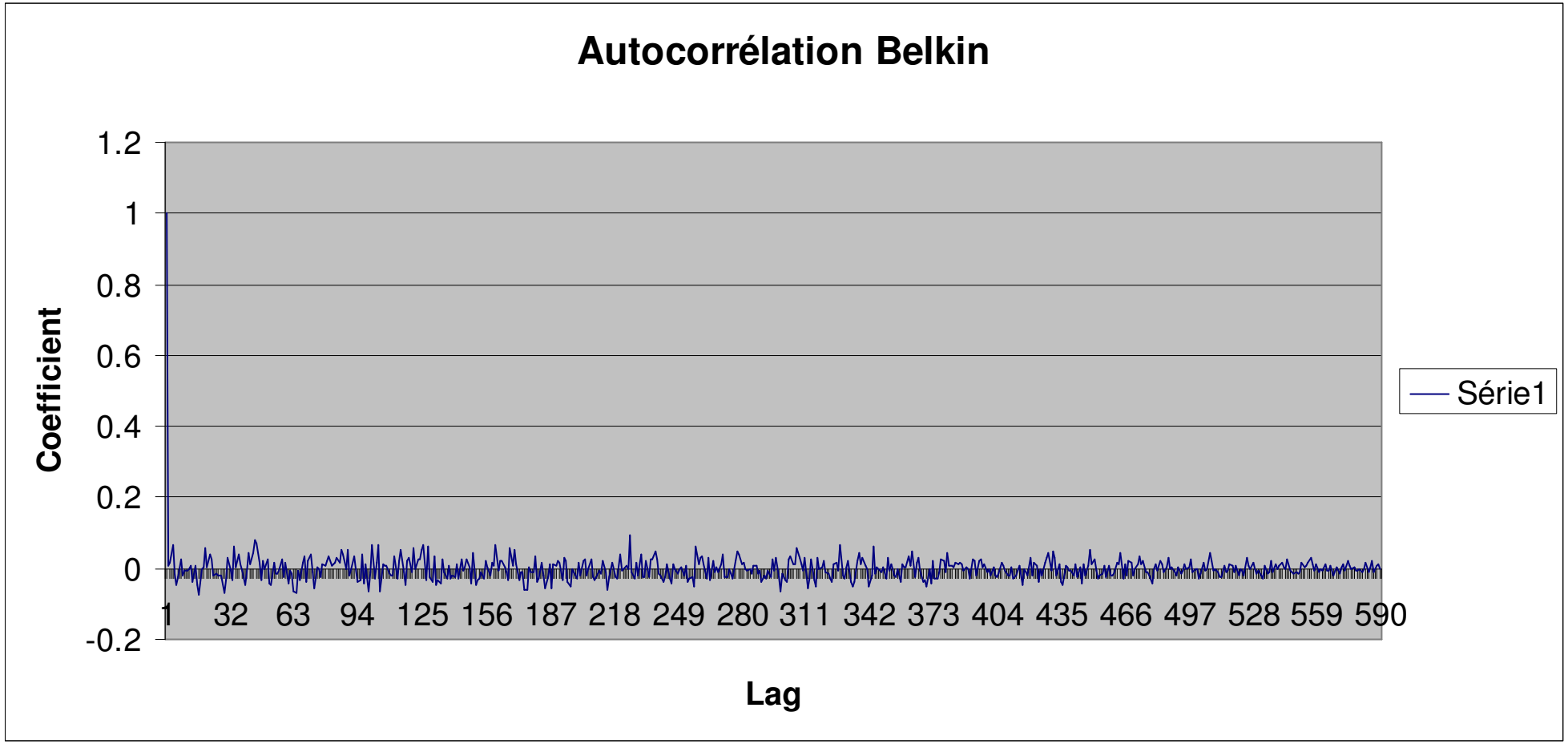

Autocorrélation Mimo point d'accès (2)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
1
199
397
595
793
991
1189
1387
1585
1783
1981
2179
2377
2575
2773
2971
Lag
Série1

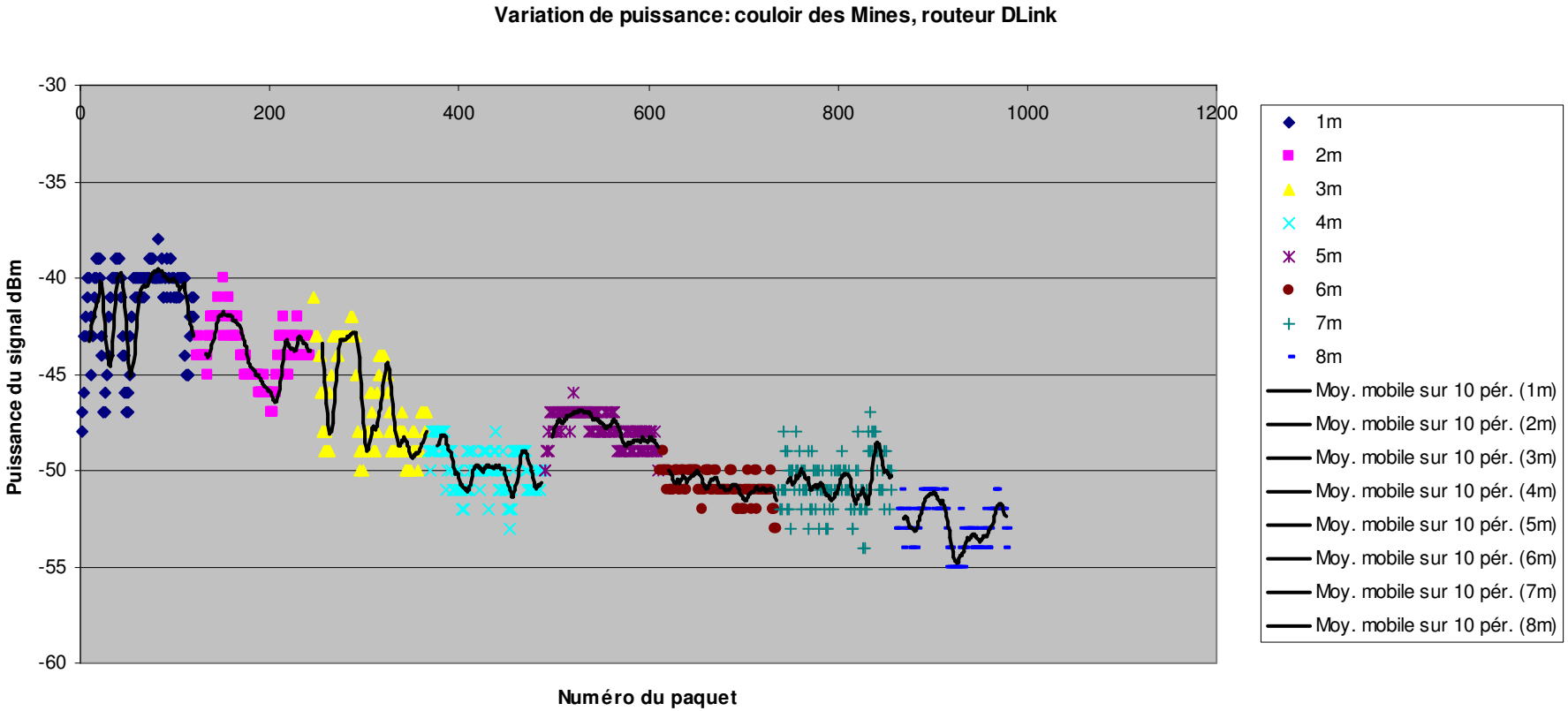

Autocorrélation Mimo point d'accès + routeur (1)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
1 34 67 100 133 166 199 232 265 298 331 364 397 430 463 496 529 562 595 628 661 694
Lag
Série1

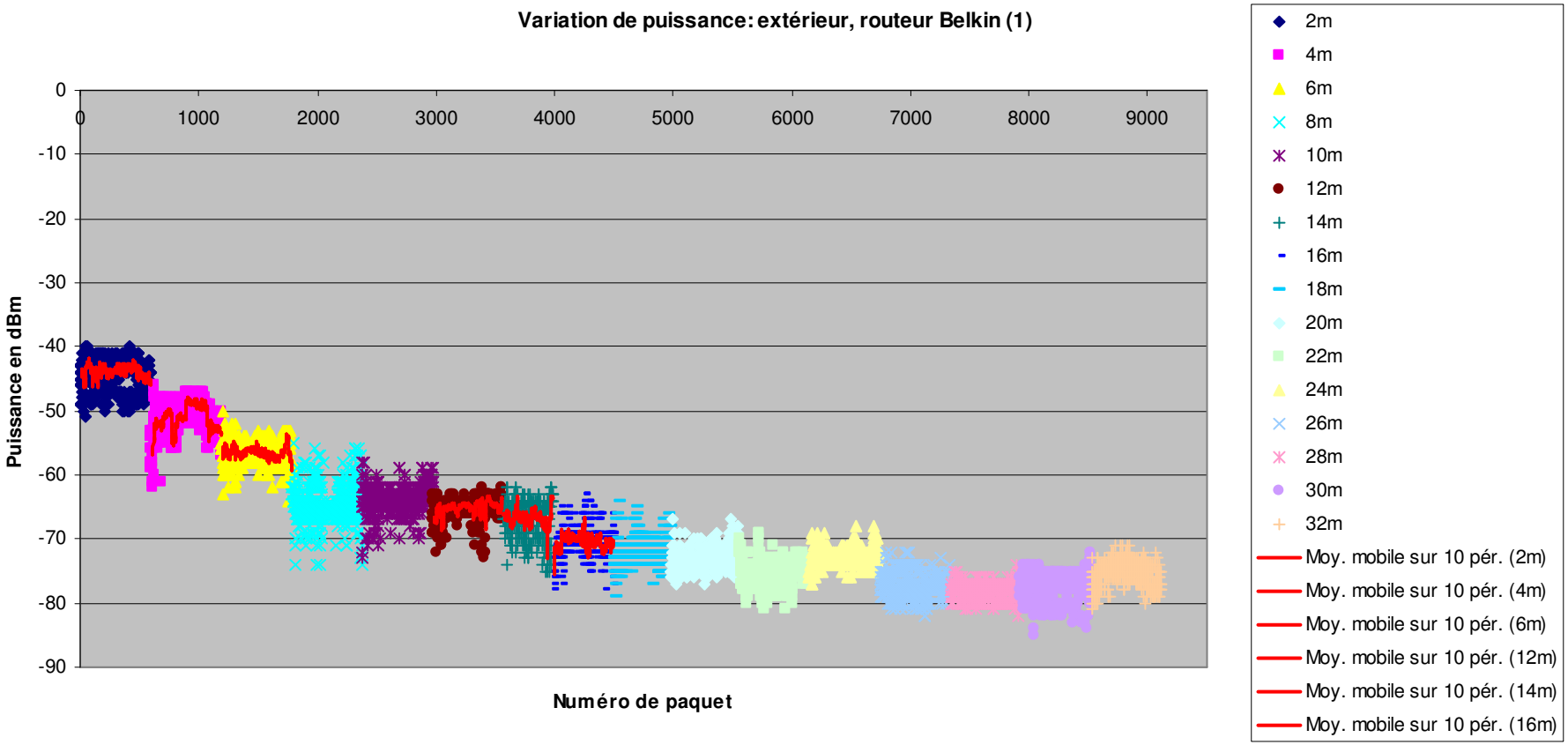

Autocorrélation Mimo point d'accès + routeur (2)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
1 39 77 115 153 191 229 267 305 343 381 419 457 495 533 571 609 647 685
Lag
Série1

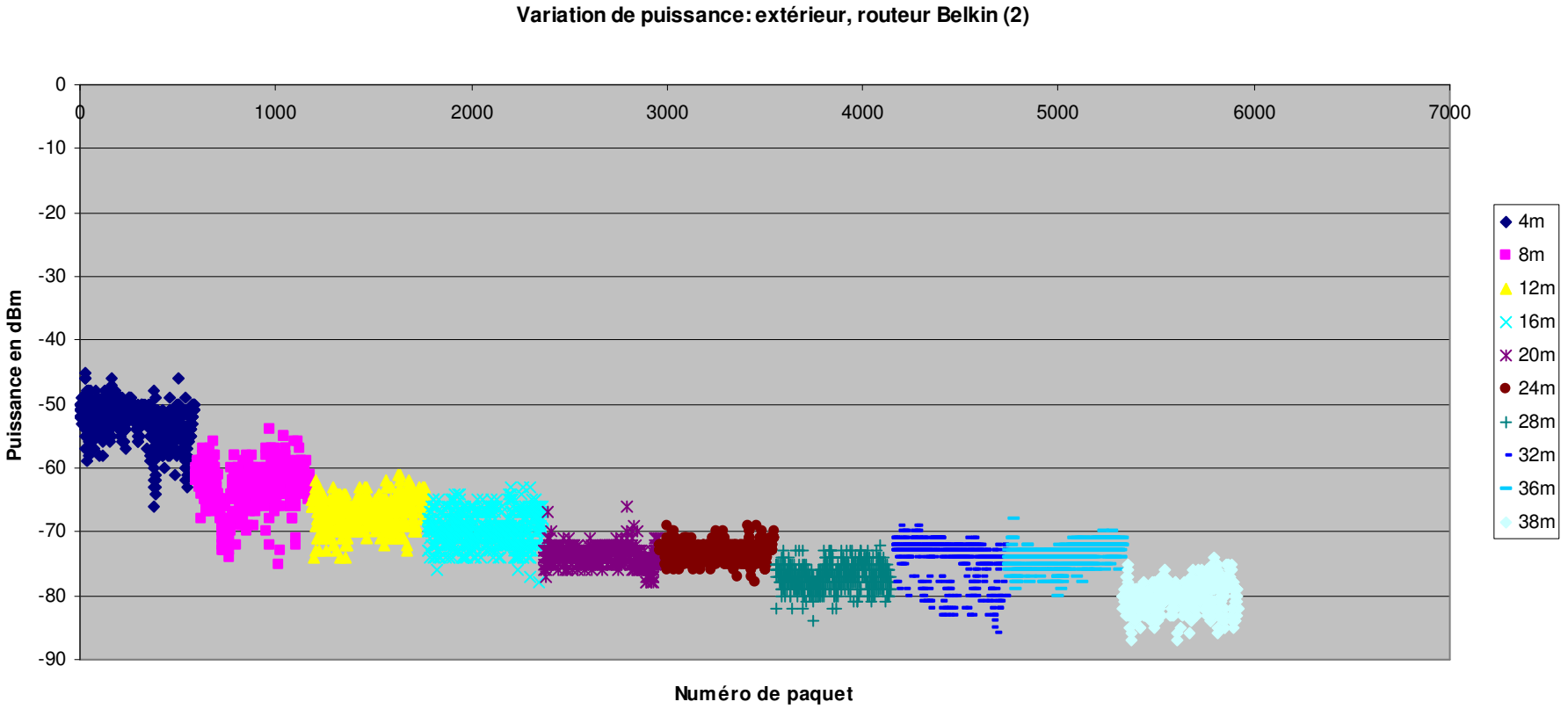

Autocorrelation DLink (1)
Coefficient
1,2
1
0,8
0,6
0,4
0,2
0
-0,2
-0,4
-0,6
-0,8
-1
1 31 61 91 121 151 181 211 241 271 301 331 361 391 421 451 481 511 541 571
Lag
Série1

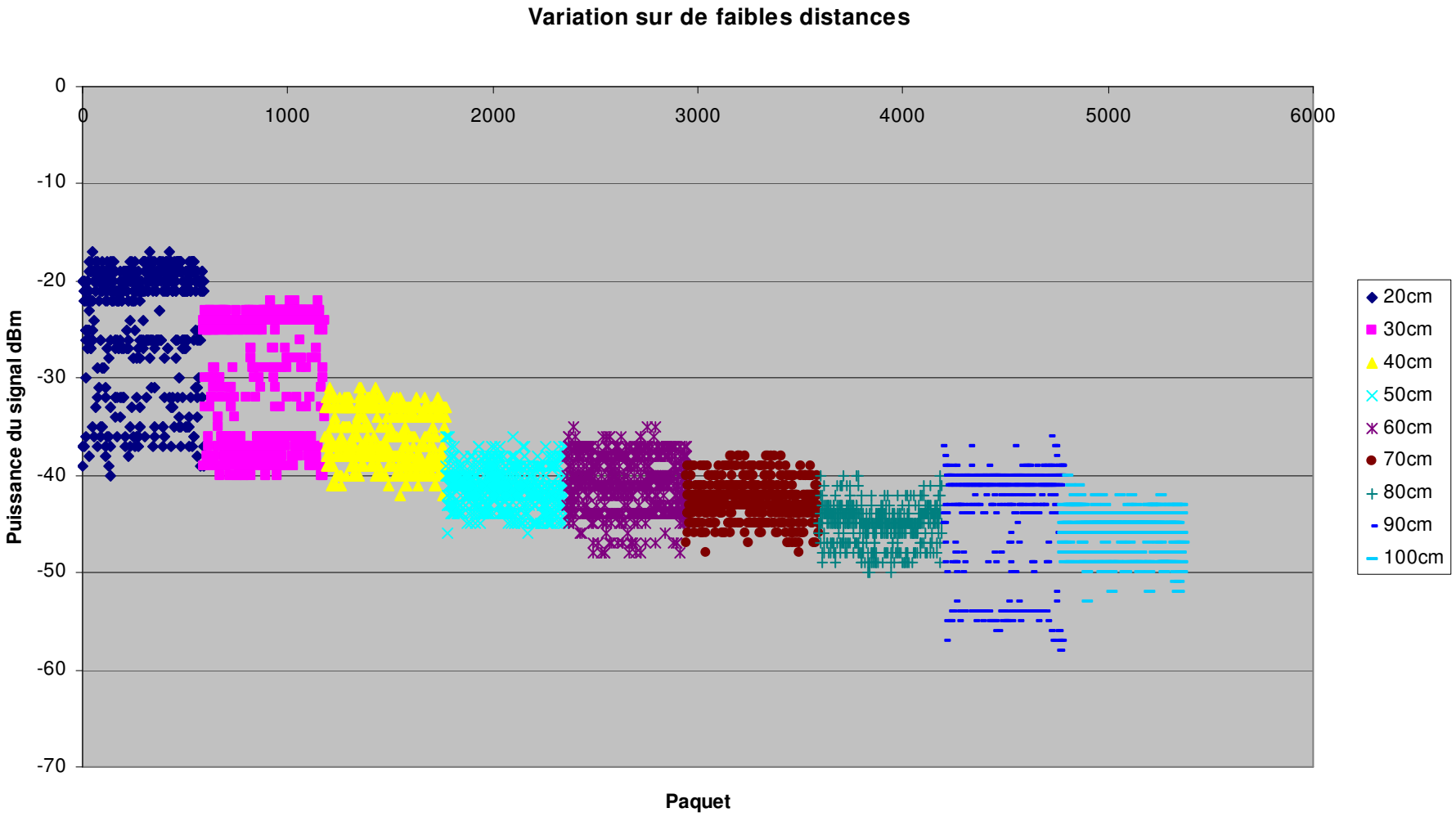

Autocorrelation DLink (2)
Coefficient
1,5
1
0,5
0
-0,5
-1
-1,5
1
16
31
46
61
76
91
106
121
136
151
166
181
196
211
226
241
256
Lag
Série1

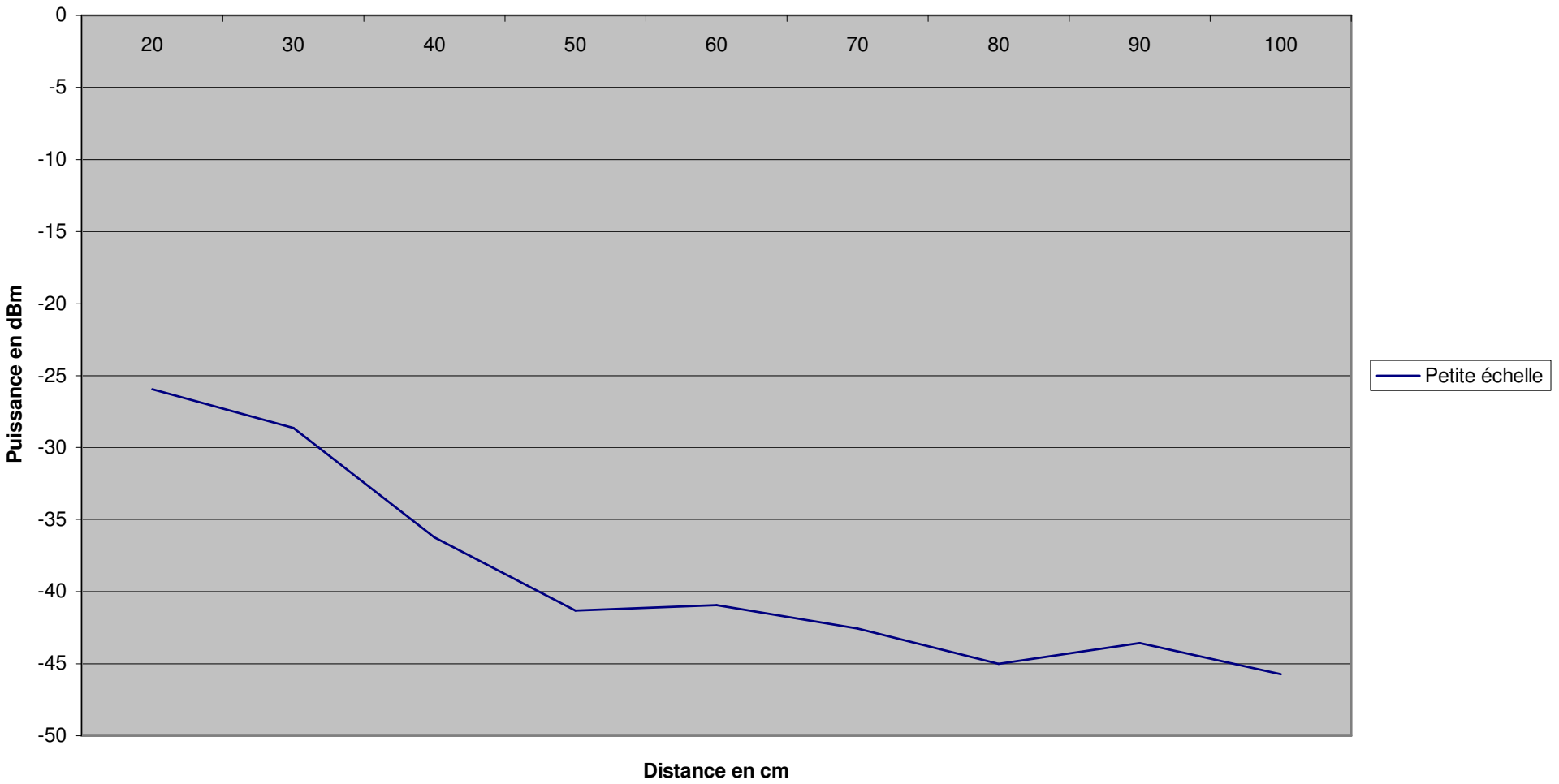

Autocorrélation Belkin
Coefficient
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
1
32
63
94
125
156
187
218
249
280
311
342
373
404
435
466
497
528
559
590
Lag
Série1

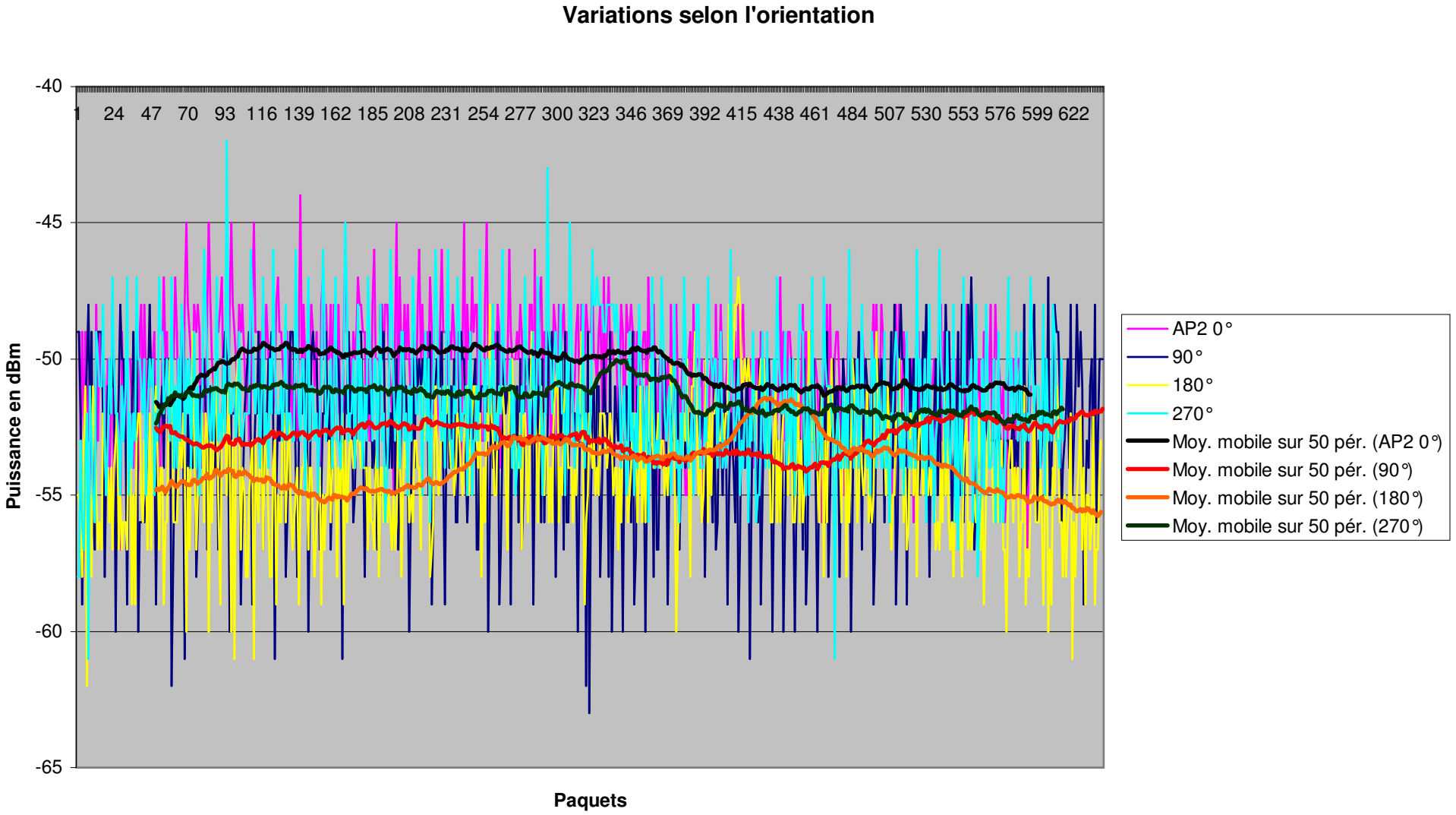

Variation de puissance: couloir des Mines, routeur DLink
Puissance du signal dBm
-30
-35
-40
-45
-50
-55
-60
0
200
400
600
800
1000
1200
Numéro du paquet
1m
2m
3m
4m
5m
6m
7m
8m
Moy. mobile sur 10 pér. (1m)
Moy. mobile sur 10 pér. (2m)
Moy. mobile sur 10 pér. (3m)
Moy. mobile sur 10 pér. (4m)
Moy. mobile sur 10 pér. (5m)
Moy. mobile sur 10 pér. (6m)
Moy. mobile sur 10 pér. (7m)
Moy. mobile sur 10 pér. (8m)


Variation de puissance: extérieur, routeur Belkin (1)
Puissance en dBm
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
Numéro de paquet
2m
4m
6m
8m
10m
12m
14m
16m
18m
20m
22m
24m
26m
28m
30m
32m
Moy. mobile sur 10 pér. (2m)
Moy. mobile sur 10 pér. (4m)
Moy. mobile sur 10 pér. (6m)
Moy. mobile sur 10 pér. (12m)
Moy. mobile sur 10 pér. (14m)
Moy. mobile sur 10 pér. (16m)

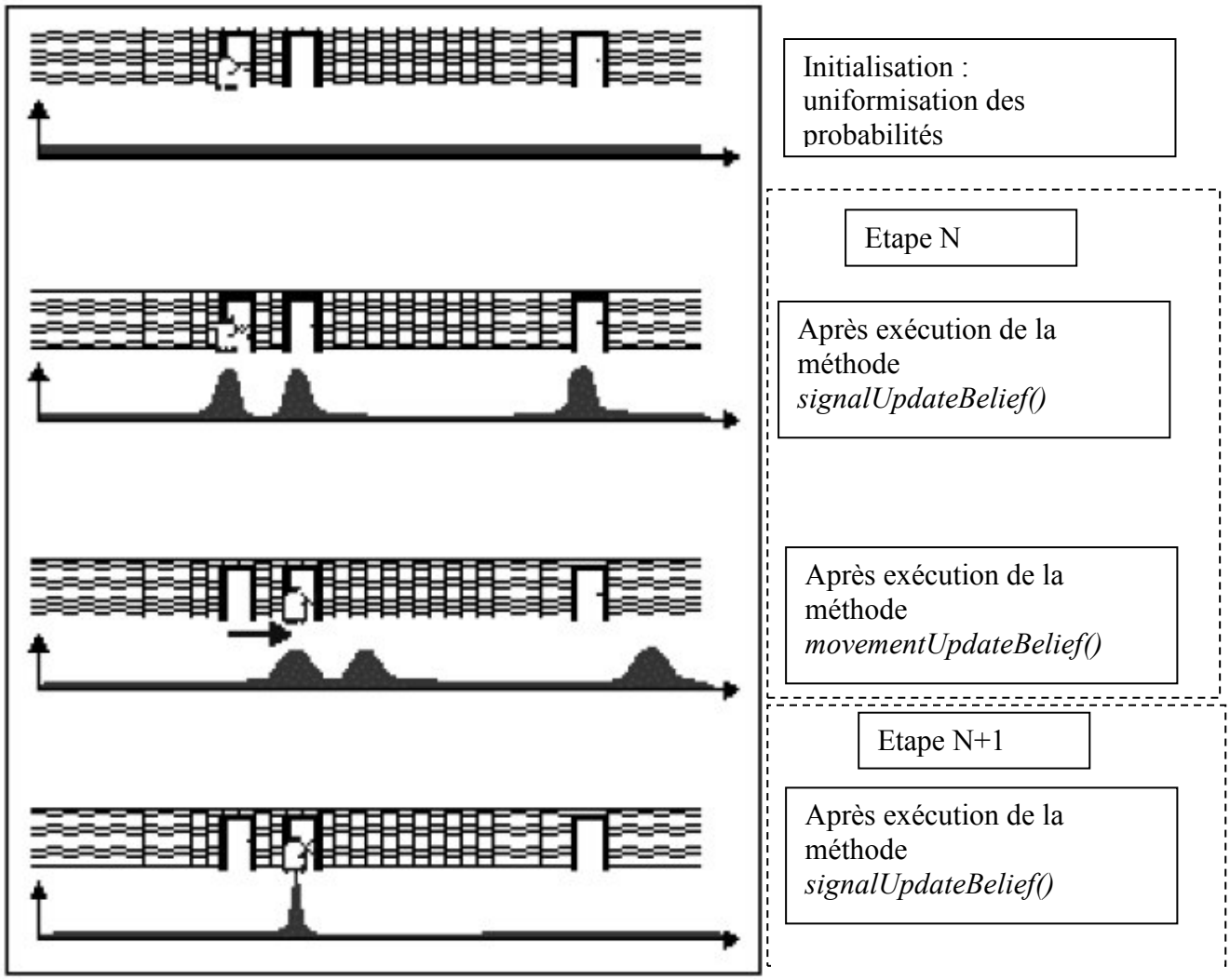

Variation de puissance: extérieur, routeur Belkin (2)
Puissance en dBm
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0
1000
2000
3000
4000
5000
6000
7000
Numéro de paquet
4m
8m
12m
16m
20m
24m
28m
32m
36m
38m


Variation sur de faibles distances
Puissance du signal dBm
0
-10
-20
-30
-40
-50
-60
-70
0
1000
2000
3000
4000
5000
6000
Paquet
20cm
30cm
40cm
50cm
60cm
70cm
80cm
90cm
100cm


Moyenne à petite échelle
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
20
30
40
50
60
70
80
90
100
Puissance en dBm
Distance en cm
Petite échelle


Variations selon l'orientation
-40
-45
-50
-55
-60
-65
1 24 47 70 93 116 139 162 185 208 231 254 277 300 323 346 369 392 415 438 461 484 507 530 553 576 599 622
Puissance en dBm
Paquets
AP2 0°
90°
180°
270°
Moy. mobile sur 50 pér. (AP2 0°)
Moy. mobile sur 50 pér. (90°)
Moy. mobile sur 50 pér. (180°)
Moy. mobile sur 50 pér. (270°)

2. Illustration simpliste unidimensionnel du principe de filtre de Bayes



3. Code source du sniffer

- **Version *offline***

```
#include <stdio.h>
#include <pcap.h>
#include "iwlib.h"


void
callback(	u_char *user,
		const struct pcap_pkthdr *h,
		const u_char *buff)
{


	FILE *file = (FILE *)user;

	//print_packet(buff,h->len);
	printf("\nPaquet de taille [%d] capturé\n", h->len);
	printf("Taille du prism: %d\n",sizeof(wlan_ng_prism2_header));
	printf("Taille du header ieee: %d\n",sizeof(struct mgmt_header_t));
	printf("Taille du corps ieee: %d\n",sizeof(struct mgmt_body_t));

	wlan_ng_prism2_header *sniff_prism;
	struct mgmt_header_t  *sniff_ieee_header;
	struct mgmt_body_t    *sniff_ieee_body;


	//Vérification de lecture des entêtes selon la taille de la frame obtenue
	if( h->len < sizeof(wlan_ng_prism2_header)){
		fprintf(stderr,"Impossible de lire l'entete prism\n");
		return ;
	}else{
		sniff_prism = (wlan_ng_prism2_header *)(buff);
	}
	if( (h->len - sizeof(wlan_ng_prism2_header)) < sizeof(struct mgmt_header_t)){
		fprintf(stderr,"Impossible de lire l'entete IEEE802.11\n");
		return ;
	}else{
		sniff_ieee_header  = (struct mgmt_header_t *)(buff+sizeof(wlan_ng_prism2_header));
	}

	/*FILTRAGE: On ne prend que les beacons pour le moment, pour cela le frame control doit valoir 128
	if( sniff_ieee_header->fc != 128 ){
		return ;
	}*/



	//L'adresse MAC de l'AP qui envoie le beacon
	int i;
	for( i = 0; i< 6 ; i++){
		if( i< 5){
			fprintf(file,"%x:",*(sniff_ieee_header->sa+i));
		}else{
			fprintf(file,"%x ; ",*(sniff_ieee_header->sa+i));
```

```
                }
        }


        //Signal
        u_int32_t signal = sniff_prism->signal.data;

        //Mactime
        u_int32_t mactime = sniff_prism->mactime.data;

        fprintf(file,"%d ; %d ; \n",signal,mactime);

        //On vide le buffer
        fflush(file);
}



int
main(   int argc,
        char *argv[] )
{

        char *x,*y;

        //Analyse de la commande et ouverture du fichier pour dumper
        if(argc == 4){
                x = argv[2];
                y = argv[3];
        }else{
                printf("Syntaxe: Sniffer  interface positionX positionY \n");
                return -1;
        }


        FILE *fp;
        char basename[5];
        sprintf(basename,"x%sy%s",x,y);


        if((fp = fopen(basename,"a")) == NULL){
                printf("Impossible d'ouvrir le fichier pour la sauvegarde");
                return -1;
        }

        char *test = "hello";


        const int snaplen = 1000;
        const u_char *packet;

        char *dev = argv[1];
        char errbuf[PCAP_ERRBUF_SIZE];

        pcap_t *handle;

        /*Strutures*/
        struct pcap_pkthdr header;
```

```
	if((handle = pcap_open_live(dev,snaplen,1,1000,errbuf)) == NULL ){
		fprintf(stderr,"Impossible d'acceder à l'interface %s: %s\n",dev,errbuf);
		return -1;
	}

	char *user = (u_char *)fp;//NULL;
	printf("%d %d\n",fp, user);



#if 1

	if( pcap_loop(handle,-1,callback,user) < 0 ){
		fprintf(stderr,"pcap_loop: %s\n",pcap_geterr(handle));
	}
#endif

	return(0);

}
```

- **Version *online***

```
#include <stdio.h>
#include <pcap.h>

#include <unistd.h>

#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>

#include <sys/types.h>
#include <sys/socket.h>

#include "iwlib.h"
#include "sniffer.h"

int
main(	int argc,
	char *argv[] )
{
	const int snaplen = 1000;
	const u_char *packet;

	char *dev = argv[1];
	char errbuf[PCAP_ERRBUF_SIZE];

	pcap_t *handle;

	/*Strutures*/
	struct pcap_pkthdr header;

	int sock_server;
	int sock_client;
	struct sockaddr_in adresse;
	socklen_t longueur;

	sock_server = create_socket("localhost",999,"tcp");

	if(sock_server < 0)
		return(-1);

	listen(sock_server,1);

	int client = 0;
	while(!client){
		longueur = sizeof(struct sockaddr_in);
		sock_client = accept(sock_server, (struct sockaddr *) &adresse, &longueur);

		if(sock_client<0){
			perror("accept");
			return(-1);
		}
		client = 1;
	}

	char *user = (u_char *)&sock_client;//NULL;

	printf("nouveau client");
```

```
	if((handle = pcap_open_live(dev,snaplen,1,1000,errbuf)) == NULL ){
		fprintf(stderr,"Impossible d'acceder à l'interface %s: %s\n",dev,errbuf);
		return -1;
	}

#if 1

	if( pcap_loop(handle,-1,callback,user) < 0 ){
		fprintf(stderr,"pcap_loop: %s\n",pcap_geterr(handle));
	}
#endif

	return(0);

}


int create_socket(const char * hote,
		  const int port,
		  const char * proto)
{
	int sock;
	struct sockaddr_in adresse;
	struct hostent *hostent;
	struct servent *servent;
	struct protoent *protoent;


	/* Récupère infos sur hôte */
	if((hostent = gethostbyname(hote)) == NULL){
		perror("gethostbyname");
		return(-1);
	}
	/* Récupère infos sur protocole */
	if((protoent = getprotobyname(proto)) == NULL){
		perror("getprotobyname");
		return(-1);
	}

	/* Ouverture de la socket */
	if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){
		perror("socket");
		return(-1);
	}

	/* Définissons l'adresse de notre socket */

	memset ( &adresse,0,sizeof( struct sockaddr_in ) );
	adresse.sin_family = AF_INET;
	adresse.sin_port = htons(port);
	adresse.sin_addr.s_addr = ((struct in_addr *)(hostent -> h_addr)) -> s_addr;

	if(bind(sock,(struct sockaddr *) &adresse, sizeof(struct sockaddr_in)) < 0){
		close(sock);
		perror("bind");
		return(-1);
	}

	return (sock);
```

```
}

void
callback(           u_char *user,
                    const struct pcap_pkthdr *h,
                    const u_char *buff)
{
        char trace[100] = "";
        char buffer[50] = "";

        wlan_ng_prism2_header *sniff_prism;
        struct mgmt_header_t  *sniff_ieee_header;
        struct mgmt_body_t    *sniff_ieee_body;

        int socket = *((int *)user);

        //Vérification de lecture des entetes selon la taille de la frame obtenue
        if( h->len < sizeof(wlan_ng_prism2_header)){
                fprintf(stderr,"Impossible de lire l'entete prism\n");
                return ;
        }else{
                sniff_prism = (wlan_ng_prism2_header *)(buff);
        }
        if( (h->len - sizeof(wlan_ng_prism2_header)) < sizeof(struct mgmt_header_t)){
                fprintf(stderr,"Impossible de lire l'entete IEEE802.11\n");
                return ;
        }else{
                sniff_ieee_header  = (struct mgmt_header_t *)(buff+sizeof(wlan_ng_prism2_header));
        }

        //L'adresse MAC de l'AP qui envoie le beacon
        int i;
        for( i = 0; i< 6 ; i++){
                if( i< 5){
                        sprintf(buffer,"%x:",*(sniff_ieee_header->sa+i));
                        strcat(trace,buffer);
                }else{
                        sprintf(buffer,"%x ; ",*(sniff_ieee_header->sa+i));
                        strcat(trace,buffer);
                }
        }


        //Signal
        u_int32_t signal = sniff_prism->signal.data;
        //Mactime
        u_int32_t mactime = sniff_prism->mactime.data;
        //Sequence control number
        u_int16_t seqctrl = sniff_ieee_header->seq_ctrl;

        sprintf(buffer,"%d ; %d ; %d ; \n",signal,mactime,seqctrl);
        strcat(trace,buffer);

        write(socket,trace,strlen(trace));

}
```

Bibliographie

1. Paramvir Bahl and Venkata N. Padmanabhan, “RADAR: An In-Building RF-based User Location and Tracking System” IEEE Infocom 2000, volume 2, pages 775-784, March 2000.

2. Andrew Howard, Sajid Siddiqi, and Gaurav S Sukhatme, "An Experimental Study of Localization Using Wireless Ethernet", International Conference on Field and Service Robotics (FSR'03), Lake Yamanaka, Japan.

3. A. Günther and Ch. Hoene, "Measuring Round trip Times to Determine the Distance between WLAN Nodes", Technical Report TKN-04-016, Telecommunication Networks Group, Technische Universität Berlin, December 2004.

4. Yu-Chung Cheng, Yatin Chawathe, Anthony LaMarca and John Krumm, "Accuracy Characterization for Metropolitan-scale Wi-Fi Localization," Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005), Seattle, WA, USA, June 2005.

5. Moustafa Youssef, and Ashok K. Agrawala, "The Horus WLAN Location Determination System," Third International Conference on Mobile Systems, Applications, and Services (MobiSys 2005), Seattle, WA, USA, June 2005.

6. Moustafa Youssef, and Ashok K. Agrawala, "Continuous Space Estimation for WLAN Location Determination Systems," IEEE Thirteenth International Conference on Computer Communications and Networks, Chicago, IL USA, October 11-13, 2004.

7. Moustafa Youssef, and Ashok K. Agrawala, "Location-Clustering Techniques for Energy-Efficient WLAN Location Determination Systems," International Journal of Computers and Applications, 2005.

8. Moustafa Youssef, and Ashok K. Agrawala, "Handling Samples Correlation in the Horus System," IEEE Infocom, Hong Kong, March 2004.

9. Y. Wang, X. Jia, H.K. Lee, "An Indoors Wireless Positioning System Based on Wireless Local Area Network Infrastructure," The 6th International Symposium on Satellite Navigation (SatNav 2003), Melbourne, Australia, 22–25 July 2003.

10. F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo localization for mobile robots", in IEEE International Conference on Robotics and Automation (ICRA99), May 1999.

11. Thrun, Fox, Burgard, Dellaert, "Robust Monte Carlo Localization for Mobile Robots", (2001).

12. Pierre Ficheux, "Linux embarqué, seconde édition", édition Eyrolles, septembre 2005.

13. Christophe Blaess, "Programmation système en C sous Linux", édition Eyrolles, mars 2005.